Departament de Llenguatges i Sistemes Informatics
UNIVERSITAT POLITÈCNICA DE CATALUNYA

Master in Computing

Master of Science Thesis

NIGHTLIGHTING

Imanol Muñoz Pandiella

Advisor/s: Gustavo Patow, Carlos Andújar

September 2012

Contents

List of Figures

Abstract

Global illumination techniques have briefly been addressed to urban renderings. To the best of our knowledge, the algorithm we present in this master thesis is the first one to compute nocturnal global illumination of a city in real time. As the problem of simulating nocturnal illumination of a city is to compute a high number of light sources, our technique divides the problem in two situations. First, for aerial views, we compute a rough simulation for the whole city and store it in a light map. For pedestrian views, we use a variation of traditional photon mapping technique to simulate global illumination of the user nearest subset of street lights. An integration method, inspired in the push-pull technique from hierarchical radiosity, integrates both solutions transferring information from one to the other to improve both of them. Our algorithm provides efficient rendering during walkthroughs and fly-throughs computing high quality images in real time.

Chapter 1

Introduction

1.1 Introduction

In the recent years urban rendering has become an important area in computer graphics. Urban rendering typically requires hundreds of thousands of buildings, each one with its own details and characteristics. Its rendering is a key ingredient in many applications, from urban planning and architectural design, to video games and entertainment. Some of these applications require not only to render realistic and detailed urban spaces, but also to perform well in real-time and to improve image quality with some kind of illumination. Because of this, real-time urban rendering with realistic illumination is still a challenging problem in computer graphics.

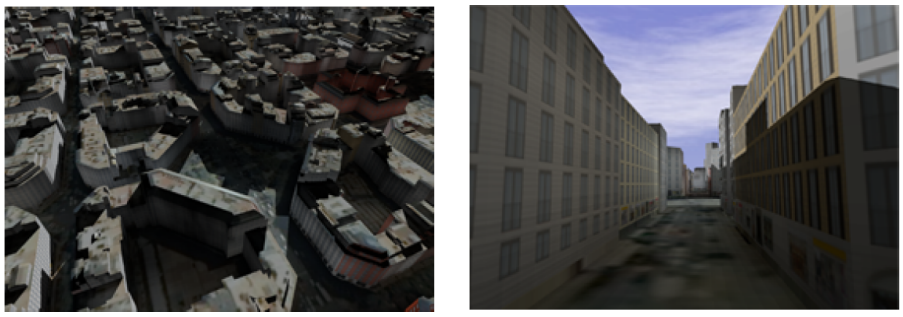


Figure 1.1: Urban rendering with global illumination. Images from [3].

Detailed urban models are often represented as textured polygonal meshes which provide a high-quality representation at the expense of a high rendering cost. Polygonal meshes are suitable for renderings involving a relatively small number of buildings, but not for large-scale urban rendering, as the rendering cost of each building is proportional to the complexity of its polygonal representation. For this reason, a number of techniques have been proposed to accelerate rendering of urban spaces. Besides view-frustum and occlusion culling techniques, related work has focused mainly on providing

level-of-detail (LOD) representations so that buildings located far away from the viewpoint are rendered in a more efficient way with little or no impact on visual quality of the resulting images.

Global illumination in urban spaces has been briefly addressed in the literature. Typically, global illumination computations have been oriented to relatively small areas. When computing global illumination, high-quality images are obtained using e.g. complete ray-tracing algorithms although the expensive cost of this kind of computations involves, even for not highly complex scenes, off-line rendering. For this reason, most of the techniques proposed to achieve real-time rates are based on approximating illumination computations minimizing the visual impact on the results. The typical solution consists to reduce the number of rays traced and introducing some kind of shapes or volumes to estimate light.

1.2 Objectives

To date, a number of algorithms have been published related to the illumination computation of generic scenes, but almost nothing has been studied with respect to global illumination computations in an environment such as modern cities. For this reason, the objective of this project is to create algorithms for the efficient computation of the nocturnal illumination of a city.

The problem is complex, as urban nocturnal illumination involves, simultaneously, a very high number of light emitters. Fortunately, there are several aspects that can reduce the computational complexity of the algorithm. For example, observers would not be able to view all the light emitters simultaneously unless they are flying over the city and, in that situation, details would not be perceptible, so a high level of approximation is possible. Furthermore, when the observer is on the streets, a small number of lights would be observed but, in this case, a high level of detail would be needed to correctly appreciate the interaction of lights and shadows with the geometry of the scene. This balance should be achieved with hierarchical structures.

The idea proposed in this thesis is to use a hierarchical structure to represent the different views and refine the quality of the information as needed, maintaining the consistence between each level.

1.3 Contributions

In order to achieve our goal, we propose to first handle light simulation for aerial views. To do this, we propose a two-pass algorithm to render this approximated version. In the first pass, we build a light map of the floor of the city drawing textured quads at the light emitter locations taking advantage

of the actual pipeline of graphic cards. In the second pass, taking advantage of the capacity of graphic cards to handle alpha blending operations, we color the scene using the light information stored in the light map. This process will produce an approximate view, but with enough quality for the considered scale. See figure 1.2.

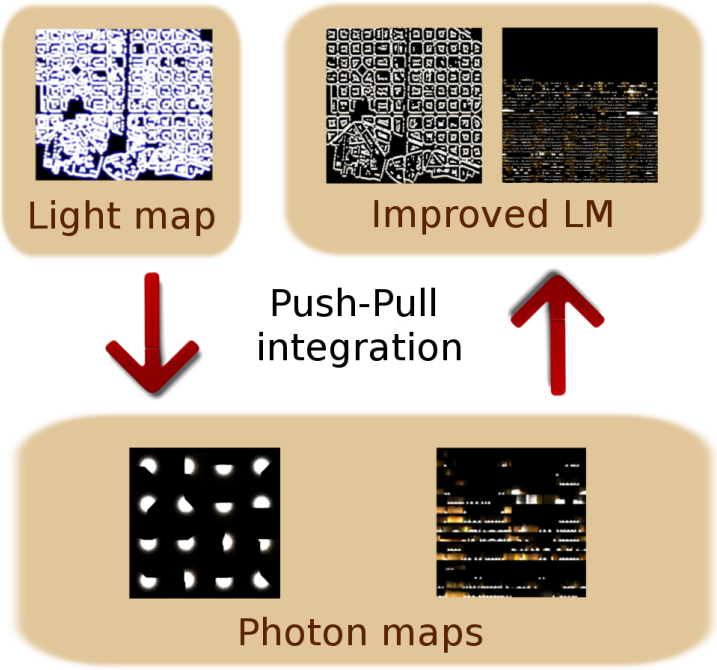


Figure 1.2: An overview of the proposed algorithm.

Subsequently, for nearby views, we propose to use a specifically tailored Photon Mapping technique based on the *Optix* engine to compute the reflexions and inter-reflexions of photons emitted from street lights. Each light emitter represents a street light. The distribution of the emitted illumination is equivalent to the distribution of a real street lamp. To store the photon information, we propose to use a data structure designed specifically with that objective in mind, which is based on 2D textures. This allows a fast access to the information during the visualization step and a compact and efficient storage.

Finally, to maintain coherence between the data structures and, therefore, between the different views, we propose, each time a photon map is computed, to correct the information of the first step. This process is very similar to the well known process of push-pull used in hierarchical radiosity [7].

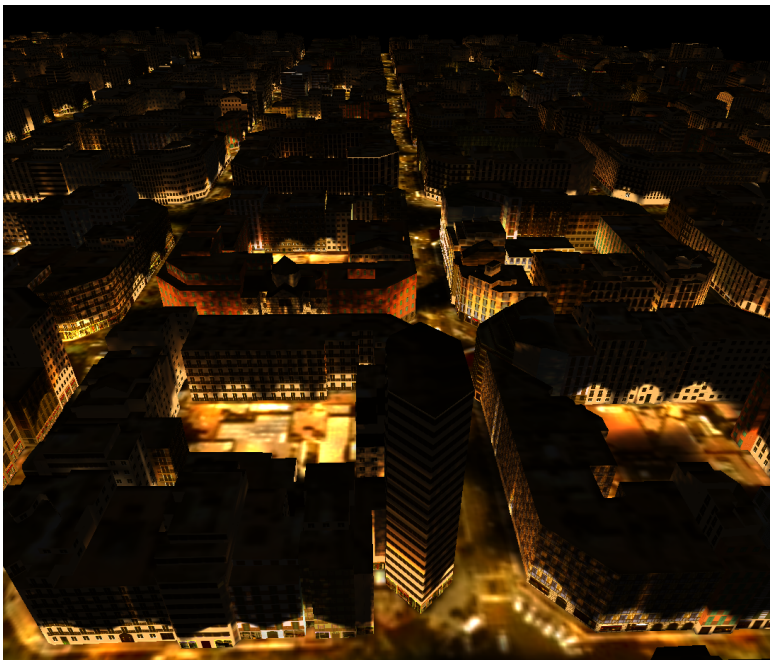


Figure 1.3: View of the night lighting of a city using our method.

1.4 Organization

The rest of the work is organized as follows. In the next chapter, we discuss the state of the art in interactive global illumination techniques, real-time city rendering and urban photon mapping. In Chapter 4 we make an overview of the presented solution, explaining how we have subdivided the problem and how we have solved each part of them and their integration. Subsequently, the results of all the steps of the solution and of the final integration are presented on Chapter 5. See figure 1.3. Finally, in Chapter 6 we present conclusions and future work.

Chapter 2

State of the Art

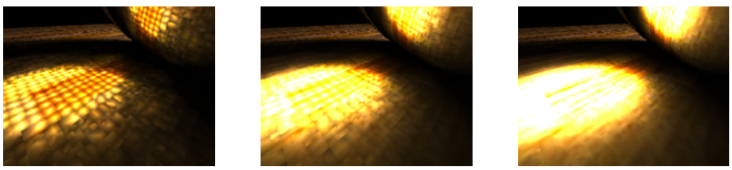
The problem of the illumination of an urban environment at night has not been addressed in depth in the literature. On the other hand, several techniques in global illumination using photon mapping have been proposed in last years. Furthermore, there have been remarkable advances in urban rendering. For these reasons, in order to understand the current state of the art in urban rendering with photon mapping, in the next chapters we briefly discuss the state of the art of global illumination and of urban rendering separately. We also review a method that, to the best of our knowledge, is the first to combine both aspects.

2.1 Interactive global illumination

We first present the origin of photon mapping. Then we discuss some important techniques which allowed interactive ray tracing. After that, we introduce the concept of splatting and how it has been an influence in global illumination techniques. Finally, we present the illumination part of the urban photon mapping technique.

2.1.1 Photon Tracing

Before discussing the different alternatives of interactive rendering of scenes with global illumination, we should take a look at the seminal paper of Jensen [14] in which photon mapping was introduced. This technique consists on a two-pass method which decouples illumination from visibility. In that process photons are emitted from the light sources and stored in two photon maps: one of high resolution for caustics and another of low resolution for general purposes. To render the final image, Monte Carlo ray tracing is used. Direct illumination is computed throwing shadow rays to light sources. Specular reflection are computed with a limited number of



(a) 0.5s (b) 1.0s (c) 2.0s

Figure 2.1: A detailed image of the GLASS BALL caustic over time obtained with [22]. Reasonably high quality estimates are available much sooner than a fully converged solution. Image from [22].

rays. To approximate soft indirect illumination and caustics, the information stored in the photon maps is used to gather the photons nearby.

2.1.2 Interactive Ray Tracing

Fast geometric ray tracing approaches have received considerable attention in the past few years. Among the most relevant once, we can distinguish the seminal paper of Wald et al. [26]. In that paper, techniques were developed to speed up the ray intersection computations, the scene traversal and shading. The main contribution of these techniques was to exploit the coherence of primary and shadow rays by traversing, intersecting, and shading packets of rays in parallel. Using this approach, it is possible to reduce the compute time of the algorithm by using SIMD instructions on multiple rays in parallel, reduce memory bandwidth by requesting data only once per packet, and increase cache utilization at the same time.

After Wald et al.'s contribution, much research has been done in that area, and many variants have been proposed to continue increasing the performance of the construction and traversal of acceleration structures for

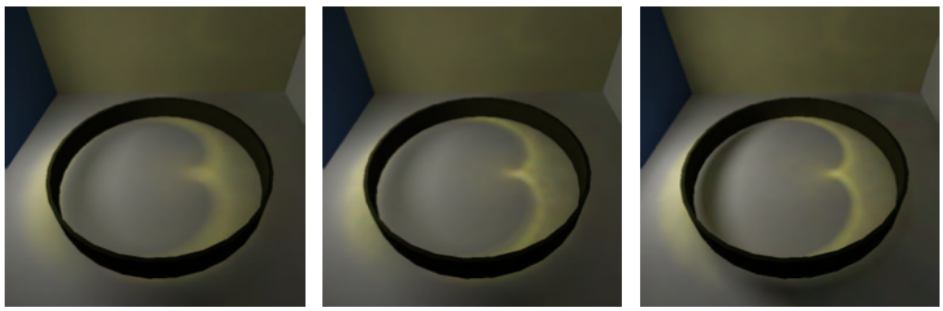


Figure 2.2: Three images rendered with the photon mapping technique [18]. Left: fixed bandwidth; Middle: adaptive bandwith; Right: reference solution. Image from [18].

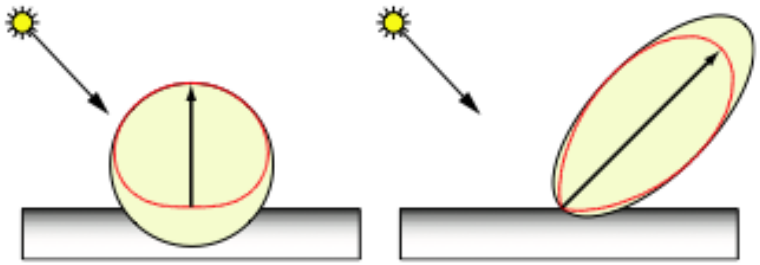


Figure 2.3: In [9] the bounding geometry for secondary light sources it is adapted to reflection properties of the surface. The left image shows a diffuse, the right image a glossy reflection. Image from [9].

GPU ray tracing. Aila and Laine [1] found that the reasons of the success or failure of any GPU tracing method were poorly understood. Focused exclusively on the efficiency of trace operations on GPUs (acceleration structure traversal and primitive intersection) and after comparing the results of optimized variants of some of the fastest GPU tracers against a custom simulator, they discovered that these kernels are below the optimum performance (a factor of 1.5-2.5) due to unbalanced hardware work distribution. In their article they showed that the performance of that kernels can be improved significantly by relying on persistent threads instead of the hardware work distribution mechanisms.

Focusing on global illumination techniques and, more specifically, on photon mapping methods, the first GPU implementation of photon mapping was developed by Purcell et al. [22]. The first step consists on doing a breadth-first photon tracer to distribute the photons using the GPU. The photons are stored in a grid-based photon map using one of two methods: the first method is a technique that uses fragment programs to directly sort the photons into a compact grid. The second method uses a single rendering pass combining a vertex program and the stencil buffer to route photons to their respective grid cells, producing an approximate photon map. To estimate the radiance, they developed a kNN-grid method to gather the photons that consists on an algorithm for finding the k-nearest neighbors to a sample point in a uniform grid. This search it is performed looking in the cell of the query point. A photon is added to the radiance estimate if it is inside a maximum search radius. The surrounding cells are considered as the search radius increases. This process finishes when the number of photons added is not less than the number requested.

2.1.3 Splatting Global Illumination

Splatting is a point-based rendering method that projects a sample onto the screen. Global illumination splatting for offline rendering was introduced by Sturzlinger and Bastos [24] using the stencil buffer to identify surfaces and with the surface BSDF assumed to be constant during splatting.

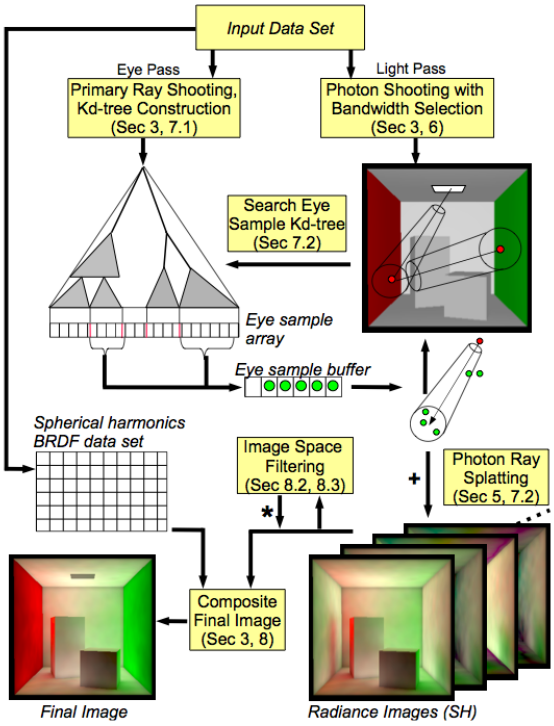


Figure 2.4: The processing flow in the photon splatting architecture [13]. Image from [13].

Subsequently, Lavignotte and Paulin [18], were the first to use a GPU on the evaluation of a splatting global illumination algorithm. The algorithm consists of a two-pass method where, on the first pass, photons are emitted from all the light sources and it is evaluated if a photon is absorbed or reflected according to a BRDF using russian roulette. During the second pass, the radiance is estimated introducing a series of approximations like a constant kernel, grouping similar photons and simplifying this process to fit hardware and avoid data dependences. The most significant contribution is the method for correcting boundary bias resulting from the constant kernel. This contribution consists of an adaptive solution to refine the result based on the correctness of the pixels computed with an heuristic function.

Another use of splatting in global illumination was introduced by Dachsbacher and Stamminger modifying the way to compute Reflective Shadow Maps [8]. This technique, developed by the same authors, consists of consid-

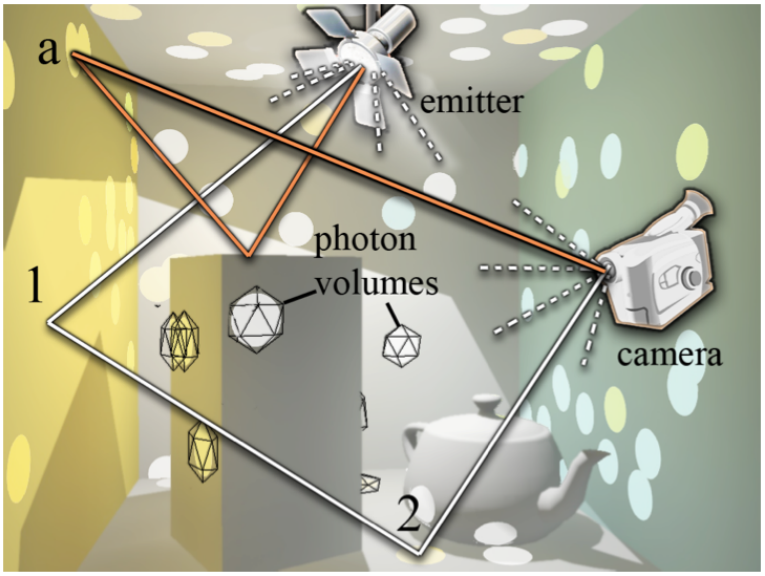


Figure 2.5: ISPM [20] algorithm. It is highlight a few ISPM light transport paths (lines), photons (discs), and photon volumes (wireframe) in a rendered scene. Line path segments from emitters and to the camera have a common center of projection, so they can be computed with rasterization on the GPU. Image from [20].

ering all the points in a shadow map as once-bounce indirect illumination. To render the final image, the once-bounce points are considered as virtual point lights and the radiance estimation is done by gathering these points. In their modification [9], they accelerated Reflective Shadow Maps by splatting light on the scene and they achieved to increase the quality of the caustics obtained by adapting the light's splat shape to the surface glossiness.

One variation of the splatting technique that increases the quality of the image obtained without using more photons was presented by Herzog et al. [13]. The algorithm consists of doing a first pass from the eye to obtain the eye positions samples and, when this pass finishes, the light pass starts with photon sampling. In the next phase all photons are splatted one by one to the eye as samples in the vicinity of the photon ray using a density estimation kernel. The final image is composed from the radiance. One of the contributions of this technique was that it splats 3D volumes instead of 2D volumes, as all previous techniques do.

One of the most interesting techniques of splatting global illumination is Image Space Photon Mapping (ISPM). It was presented by McGuire and Luebke [20] and it allows global illumination for real-time applications in high definitions resolutions, as well as accelerating it for offline rendering. The algorithm starts emitting photons via the GPU and tracing them con-

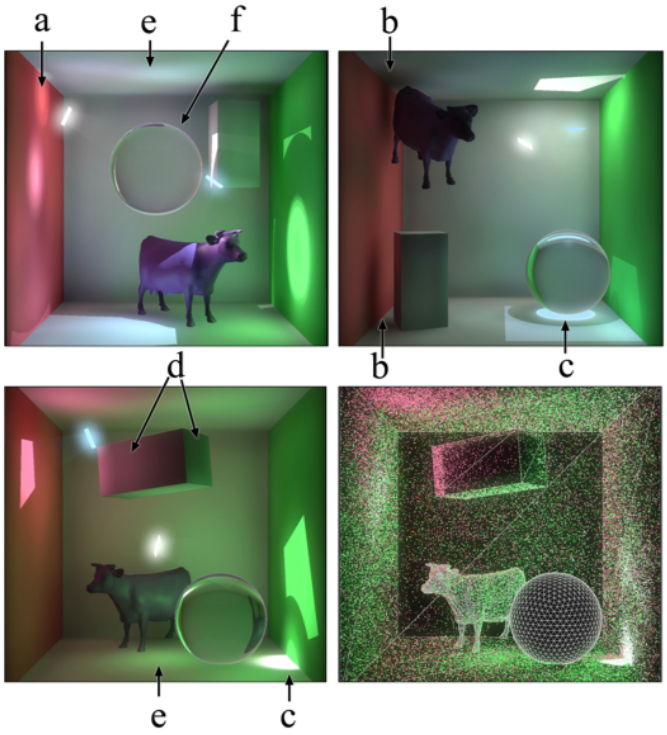


Figure 2.6: ISPM [20] global illumination effects: (a) reflective caustic, (b) contact shadows, (c) refractive caustics, (d) diffuse interreflection, (e) refractive caustic from indirect illumination and (f) Fresnel reflection. Image from [20].

ventionally at the CPU. The key step comes with radiance estimation, ISPM scatters indirect illumination by rasterizing an array of photon volumes. These photon volumes are an extension from the 3D splats of Herzog et al. [13], which conform more tightly to surfaces and avoid the expensive cone estimation step. The contribution of ISPM is that it is a simple unified solution to all illumination phenomena, obtaining high quality results in real time for high definition resolutions.

2.2 Real-time city rendering

In the urban rendering part, we present methods to simplify buildings and groups of buildings first. After that, it is discussed the main methods to render cities with simplification of distant buildings. Subsequently, hierarchical methods to render cities that uses relief mapping are presented. Finally, the relation between global illumination and urban rendering is discussed.

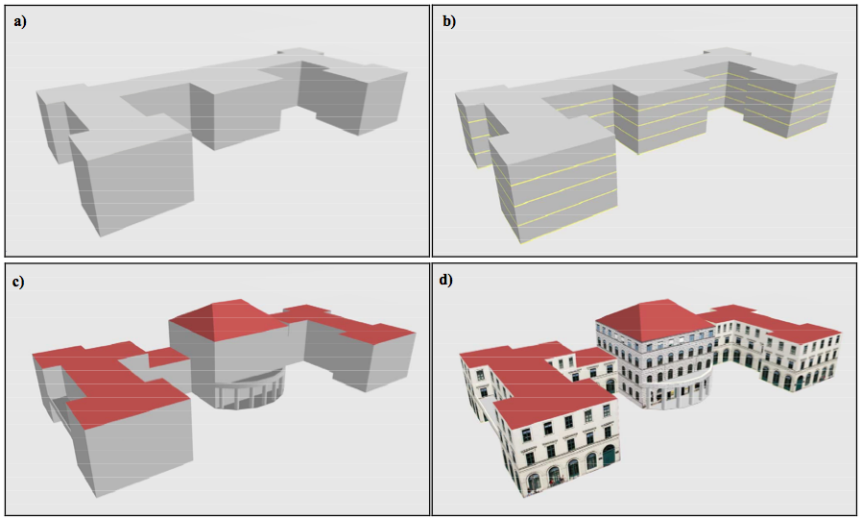


Figure 2.7: How [11] transforms a block model into a refined CLOQ building. (a) Block model, (b) CLOQ building split into floors, (c) Geometry refinement and (d) Appearance refinement. Image from [11].

2.2.1 Render of buildings

The poor performance of traditional mesh simplification algorithms on urban models has motivated the development of algorithms specifically designed for buildings and groups of buildings. One paper which focus in the problem of how to represent a building model was developed by Dällner and Buchholz [11]. In this method, the building's geometry is structured according to its floor; for each bulding its floor refers to a floor prototype, which is defined by a ground plan, walls and wall segments. To describe the appearance they used projective textures across floors and textures per wall segment. This information allowed them to efficiently express most common building features.

Subsequently, Haunert and Wolff [12] presented an optimization approach to simplify sets of building footprints represented as polygons subject to a user-defined error tolerance. Their method consists of simplifying each polygonal ring by selecting a subsequence of its original edges. The vertices of the simplified ring are defined by intersections of consecutive edges in the selected subsequence. With this simplification, they reduced the number of all output edges simplifying the buildings and, with it, the rendering cost.

2.2.2 Render of group of buildings

Chang et al. [5] realized that, although algorithms for mesh simplification based on LOD work well for simplifying single objects with a large number of polygons, they produce illegible objects when they are simple and they can

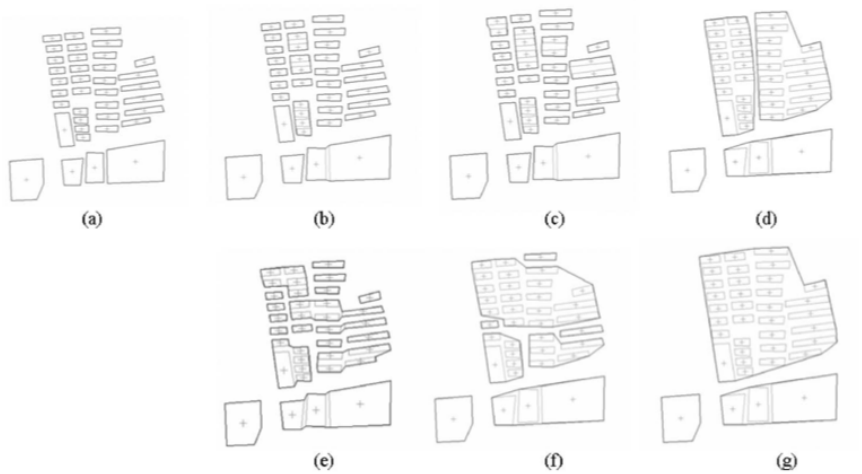


Figure 2.8: Examples of footprint simplification provided by [28]. (a) The original building footprints, (b)-(d) show the clustering results using the traditional distance, (e)-(g) illustrate the results using the adjacent distance. Image from [28].

even produce the disappearance of entire areas when the buildings tend to be much smaller than in other areas. To solve this problem, they developed a mesh-simplification algorithm based on an architectural concept called urban legibility, which segments a city into paths, edges, districts, nodes and landmarks. To do so, they based their algorithm on merging similar elements with a process divided in five steps: a hierarchical clustering during preprocessing, a cluster merging, a model simplification and a hierarchical texturing; and at runtime, it is employed LOD to select the appropriate models for rendering.

More recently, Yang et al. [28] extended the previous work including elements from the Gestalt psychology to be the basis of the mesh simplification with the urban legibility concept. The reason of adding these Gestalt psychology elements is to incorporate the way people recognize spatial relations between objects and understand urban spatial information. In that algorithm, they introduced a new distance measurement method for the clustering of buildings. After that, each cluster is merged based on the Delaunay triangulation and the polyline generalization algorithm. Then, it is constructed a hierarchical tree to store multi-resolution building models and it is designed a method to traverse the tree to achieve interactive render rates.

2.2.3 Render of distant buildings

Several image-based representations have been proposed to accelerate the rendering of distant buildings. An example of them can be impostors, textured depth meshes and point-based impostors.

Impostors

One of the first methods that tried to solve the problem of interactive rendering for walkthroughs was developed by Maciel and Shirley [19] using impostors. An impostor is an entity that is faster to draw than the true object, but retains the important visual characteristics of the true object. The key issue is how to decide which impostors to render to maximize the quality of the image without exceeding the available frame rendering time. The main properties of the technique that they presented are that they used a single hierarchy which contains drawable impostors for objects and for clusters of objects, and that the system used the graphics hardware to automatically create the hierarchy, generate the impostors, compute their rendering cost and compute an estimation of their benefit according to the direction from which they are viewed.

Textured depth meshes

One of the problems of impostors is that they can handle a limited set of scene types. For example, they typically fail for densely occluded environments. Also, due to paralax problems, the number of frames for which impostors are valid is limited. To solve this problem, the concept of textured depth meshes was created, an impostor-based scene representation that can be used to accelerate the rendering of static polygonal models.

One solution based on this idea was introduced by Sillion et al. [23] by combining the potential of impostors with the reduction of the geometric complexity of individual objects for the distant objects and adding a dynamic segmentation of the dataset. To achieve this, they presented a subdivision model of urban scenes, based on the inherent city structure, which provides a 3D model for the local neighbourhood and, for the distant objects, they used impostors augmented with 3D information allowing longer cache life and resolving the parallax problem.

More recently, another technique based on the idea of textured depth meshes was presented by Jeschke and Wimmer [17]. Their approach relies on a layered rendering of the scene to produce a voxel-based representation and the construction of a highly complex polygon mesh that covers all the voxels. Then, this new mesh is simplified using an error metric to ensure that all voxels become covered. Finally, the remaining polygons are resampled using the voxel representation to obtain their textures. The most

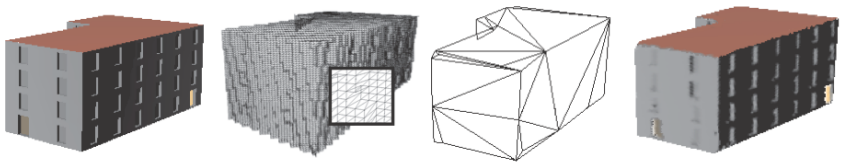


Figure 2.9: Overview of method for constructing textured depth meshes [23] (from left to right): the original object (1880 polygons), the initial mesh constructed from voxel field (61584 polygons), the simplified mesh (35 polygons), close-up view of the final textured mesh. Image from [23].

important contribution is that it can handle polygonal meshes without previous information about their structure. Also, only scene parts that may become visible are represented and the use of the error metric guarantees that impostors are almost indistinguishable compared to the original model.

Point-based impostors

Another technique based on impostors was developed by Wimmer et al. [27]. This method relies on an object-space sampled representation similar to a point cloud or a layered depth image. To improve the previous techniques based on simplifying distant geometry, they introduced some improvements. They proposed a new representation that decouples the geometry, represented as a set of 3D points, and the appearance, represented as a simplified light field computed from the original model. To achieve this, the sampling rate of the point cloud is controlled to provide enough density across all possible viewing conditions from the specified view cell. Furthermore, it is defined a proper characterization of the radiance contribution to the image from these points using a Monte Carlo integration and encoded as a texture map to model view-dependent effects.

2.2.4 Hierarchies of displacement maps

Some recent approaches represent the geometry of 2.5D cities as hierarchies of displacement maps which are rendered using relief mapping techniques.

One of the first methods based on this idea was presented by Cignoni et al. [6] who introduced the concept of a new discrete representation called BlockMap. A BlackMap represents a set of textured vertical prisms with a bounded on-screen footprint. These are stored into small fixed-size texture chunks and efficiently rendered through raycasting. In their solution, they used a visibility-aware traversal of the hierarchy to render components close to viewer with textured polygons and they employed BlockMaps for far away geometry. With this method, they presented a GPU-friendly technique that exploits the structured nature of urban environments interactively.

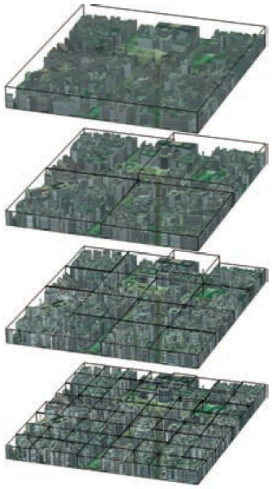


Figure 2.10: The first four levels of the BlockMaps hierarchy [6]. The same portion of the urban scene is covered with finer and finer BlockMaps arranged in a quadtree. Image from [6].

Subsequently, Di Benedetto et al. [10] extended BlockMaps to be used as direction-independent impostors when rendering far-away city blocks. Their contribution was to extend the BlockMap representation to support sloped surfaces and input-sensitive sampling of color. They introduced a novel sampling strategy for building accurate BlockMaps and they used BlockMaps to parameterize the visible surface of a highly complex model. Another important contribution was the integration of an ambient occlusion term in the rendering to improve the expressiveness of urban models and the efficient method presented to compute it.

More recently, Andújar et al. [2] presented a technique that provides a more general solution for a larger class of urban shapes as the sampling density does not depend on the perimeter of the building facades like BlockMaps do. The approach is based on a multi-resolution tree of the scene which defines multi-level relief impostors. To obtain this tree, the method precomputes a small set of zenithal and oblique relief maps that capture the geometry and appearance of the buildings inside each node of the tree. This representation is rendered through a view-dependent traversal of the tree. Rendering one node of the tree involves rendering its bounding box and the image is obtained combining relief mapping and projective texture mapping with the information of the nodes. With this method, they achieved high-quality representations, even for medium-range distances, that allow to recover facade details.

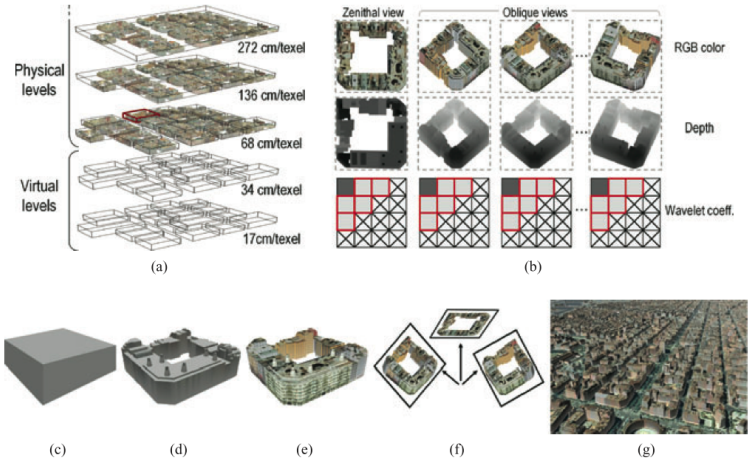


Figure 2.11: Overview of the relief impostors approach [2]. This urban model consists of a scene subdivision tree. The five deepest levels of the tree are shown in (a). The geometry and appearance of each tree node is represented by a small collection of textures (b) captured from zenithal and oblique views. The render consists of going from (c) to (g). Image from [2].

2.2.5 Global illumination

Global illumination computations have not been often used to compute the illumination in urban scenes. Indirect ilumination effects are limited to environment maps and precomputed ambient occlusion as the presented by Although Di Benedetto et al. [10] in their approach. Vanegas et al. [25] wrote a report where they point out the lack of global illumination approaches for urban models.

2.3 Urban Photon Mapping

Although there have been enormous improvements in urban rendering in the last years, the accurate photo-realistic rendering of cities has been barely touched. Argudo et al. [3] recently presented the first approach for interactive photo-realistic rendering of urban landscapes using a photon-mapping-based technique in which a large urban model is rendered in real time emulating sun lighting. They only consider a very far away light source, the sun, which emmits the photons against the city. They accumulated the photon energy into a collection of persistent 2D photon buffers encoding the income radiance for a superset of the surfaces contributing on the current image. They defined an implicit parametrization to map surface points onto photon buffer locations through a cylindrical projection for the buildings and an orthogonal projection for the terrain. To achieve real time, they used an

adaptive scheme to adapt the resolution of the photon buffers to viewing conditions that imply small temporal artifacts.

As the technique proposed in this master thesis can be considered an extension of their work with different conditions and objectives, the details of this technique are discussed deeply in Section 4.2.1.

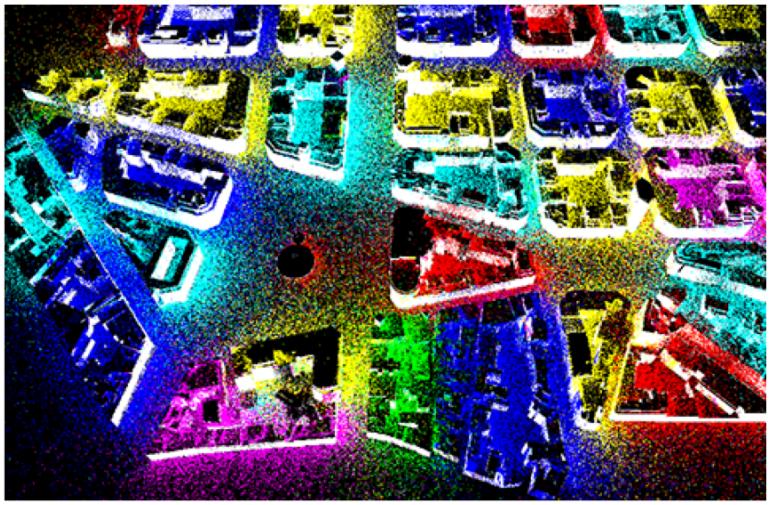


Figure 2.12: Urban Photon Mapping [3] visualization of photon hits over the scene for a Sun light located to the right at an elevation of 45 degrees. Image from [3].

2.4 Conclusions

Global illumination techniques have evolved considerably in recent years. Photon mapping is a promising technique to achieve high quality results [14]. Thanks to the improvement in ray tracing techniques to compute global illumination [26, 22, 1] it has been possible to improve splatting techniques [24, 18, 9, 8, 13] in global illumination methods until achieve real time results. Of all the actual variants of photon mapping, the most promising ones to achieve real time results are those which splat the photons instead of those which gather the radiance. In this way, the work by McGuire and Luebke [20] is the best exponent of how a technique can give a unified solution to real time and off line rendering achieving high quality results in high definition resolutions. Although this technique provides real-time results in several scenarios, it is geometry-dependent and for this reason it is not possible to use it for rendering large urban models.

Rendering of urban models has benefited from the methods to simplify buildings [11, 12] and groups of buildings [5, 28]. Several techniques have been developed to render urban models with simplified versions of distant

objects as impostors [19], textured depth meshes [23, 17] and point-based impostors [27]. More recent works using hierarchies of displacement maps [6, 10, 2] have provided real-time results rendering urban large models using relief mapping methods. It is important to mention that most of the methods that compute indirect illumination in urban scenarios [25] do it offline.

To the best of our knowledge, up to now the only exception is Urban Photon Mapping [3], which achieves real-time frame rates visualizing an urban model computing photo-realistic illumination provided by the sun. The need of the use of an adaptive version of photon mapping to achieve real time frame rates despite the temporal artifacts gives us a chance to continue researching on how to compute nocturnal lighting, with its multiple light sources, and how it would work on a urban scenario.

Chapter 3

Classic Photon Mapping

As we have previously mentioned, this master thesis is based in a global illumination technique called Photon Mapping. This approach was proposed by Henrik Wann Jensen in 1996 [14, 15, 16]. This chapter is intended as an introduction for readers, unfamiliar with this technique, going into detail on the most relevant aspects.

3.1 The Photon Mapping Concept

The photon mapping method is an extension of ray tracing capable of simulating global illumination, including caustics, diffuse interreflections and participating media. Photon mapping was developed as an efficient alternative to Monte Carlo ray-tracing. The goal was to obtain the same flexibility in a more efficient way.

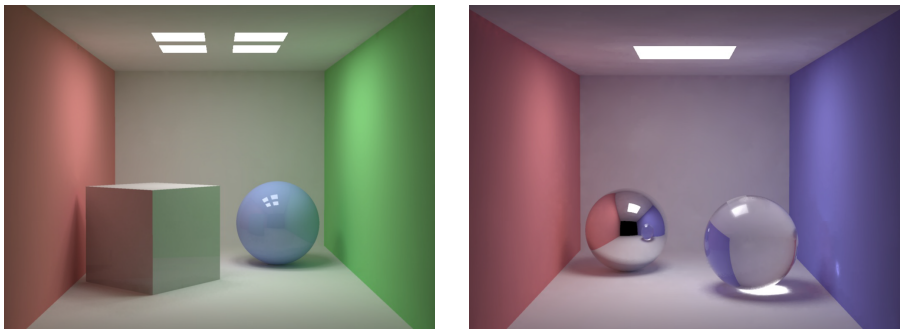


Figure 3.1: Example images generated with classic photon mapping. Figure obtained by Jensen et al. [16].

The algorithm is based on two ideas. First, to be able to handle arbitrary geometry, as well as complex models, Jensen proposed to decouple the representation of the illumination from the geometry. The second idea is to store illumination as points in a global data structure called photon map

on which radiance estimation would be a density estimation. In this way, the high-frequency noise is replaced by low-frequency noise and this estimation can be computed in a more efficient way than Monte Carlo ray-tracing methods. With this two ideas it is possible to handle complex scenes in an efficient way, but this has the problem that the average expected value of the method may not be correct. However, the technique will converge to the correct result as more photons are used.

In conclusion, the technique can be summarized in two steps. First, a photon tracing step where the photons are emitted and stored in the photon map and, after that, a radiance estimation step where the global illumination is approximated using the information stored in the photon map.

3.2 Photon Tracing

Photon tracing is the process where the photons are emitted from the light sources, traced through the scene and stored in a photon map. In the following subsections, we discuss them in detail.

3.2.1 Photon emission

Photons are created at the light sources and are emitted from it having a distribution corresponding to the distribution of the power emitted by the light source. Any kind of light sources can be used: point lights, directional lights, area light sources and more general sources with any shape and distribution.

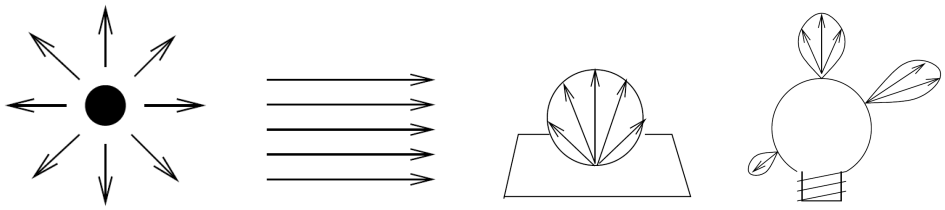


Figure 3.2: Emission from light sources: (a) point light, (b) directional light, (c) area light and (d) arbitrary light. Figure obtained from [16].

Photons from a diffuse point light are emitted in uniformly distributed random directions and, from a directional light, are emitted all in the same direction but the light source is outside the scene. To choose the directions we need to know a bounding volume that covers the whole scene. Then, a random position within the projected bounding volume is chosen and it will set the direction of the emitted photons. For area light sources, photons are emitted from random positions with random directions. These directions are

chosen from a cosine distribution to guarantee that the emission direction with highest probability is perpendicular to the square that represents the light. For arbitrary light sources the density of emitted photons must match the emission profile of the light to ensure that the power of the emitted photons is the same. One method is to chose some random positions and directions on the light and then, to scale the power of the emitted photons according to the light profile. Examples of the directions of the rays are shown in figure 3.2.

The power of the light source is divided among the photons emitted from it. If P_{light} is the power of the light source and n is the number of emitted photons, the power of each emitted photon would be:

$$P_{photon} = \frac{P_{light}}{n} \tag{3.1}$$

If the scene contains multiple light sources, photons should be emitted from each light source emitting more photons from brighter lights than from dim lights, to make the power of all emitted photons even. Although we can expect a higher number of photons must be emitted, fortunately it is not so. Each light will contribute less to the overall illumination and fewer photons are needed than in a scene with a single light source.

To speed up the computation in outdoor scenes, Jensen proposed to use projection maps to avoid emitting photons that will not hit any geometry. A projection map is a map of the geometry as seen from the light source. Each cell of the map encodes a boolean indicating if there is geometry in that direction or not. This method is a conservative solution that guarantees that photons are emitted to directions that can hit geometry.

3.2.2 Photon tracing

Once a photon has been emitted, it is traced through the scene using the photon tracing technique. Photon tracing works as ray-tracing methods, but there is an important difference. Photon tracing propagates flux instead of gathering radiance, in such a way that the interaction between photons and materials is different than the interaction of a ray. When a photon hits a surface it can be transmitted, reflected or absorbed. This decision is made probabilistically based on the material properties of the surface following the *Russian Roulette* technique. Examples of photon paths are shown in figure 3.3.

Taking into account only reflection and absorption, the technique works as follows. Considering a monochromatic simulation, for a surface with diffuse reflection coefficient d and specular coefficient s (with $d + s \leq 1$) and using a uniformly distributed random variable $\xi \in [0, 1]$, it is made the

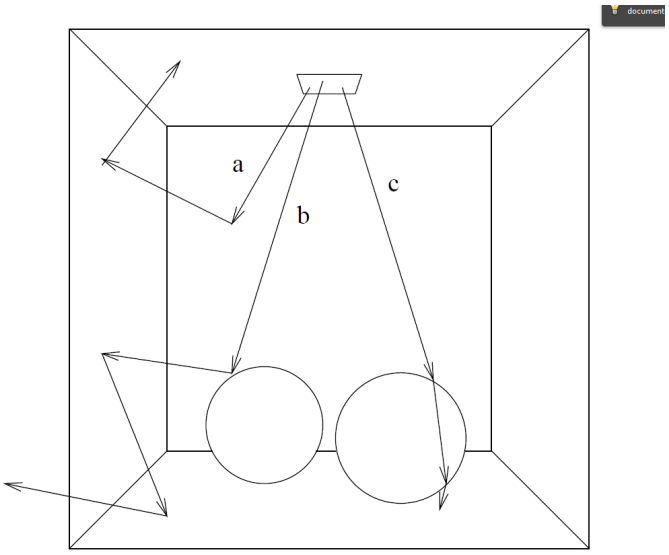


Figure 3.3: Photon paths in a Cornell Box scene with a chrome sphere on left and a glass sphere on right. (a) Two diffuse reflections and an absorption. (b) A specular reflection and two diffuse reflections. (c) Two specular transmissions and an absorption. Figure obtained from [16]

following decision:

$$\begin{aligned} \xi &\in [0, d] && \rightarrow \text{diffuse reflection} \\ \xi &\in [d, s+d] && \rightarrow \text{specular reflection} \\ \xi &\in [s+d, 1] && \rightarrow \text{absorption} \end{aligned} \tag{3.2}$$

The use of *Russian Roulette* guarantees no need to modify the power of the reflected photons because the correctness is ensured by averaging several photon interactions over time.

With more color bands the decision is similar. Considering different reflection coefficients in each color band, the probabilities of reflection can be based on the maximum energy reflected in any band. For three bands (RGB), the probabilities P_d and P_s for diffuse and specular reflection respectively can be computed as

$$P_d = \frac{\max d_r P_r, d_g P_g, d_b P_b}{\max P_r, P_g, P_b} \tag{3.3}$$

$$P_s = \frac{\max s_r P_r, s_g P_g, s_b P_b}{\max P_r, P_g, P_b} \tag{3.4}$$

where (d_r, d_g, d_b) and (s_r, s_g, s_b) are the diffuse and specular reflection coefficients in the red, green and blue color bands respectively, and (P_r, P_g, P_s) are the powers of the incident photon in the same color bands.

The probability of absorption is $P_a = 1 - P_d - P_s$. With these probabilities, the previous decision would be:

$$\begin{aligned} &\xi \in [0, P_d] && \rightarrow \text{diffuse reflection} \\ &\xi \in [P_d, P_s + P_d] && \rightarrow \text{specular reflection} \\ &\xi \in [P_s + P_d, 1] && \rightarrow \text{absorption} \end{aligned} \tag{3.5}$$

The power of the reflected photon needs to be adjusted to account for the probability of survival. For example, for specular reflection with three bands (RGB) the power P_{refl} of the reflected photon is:

$$\begin{aligned} P_{refl,r} &= P_{inc,r} s_r / P_s \\ P_{refl,g} &= P_{inc,g} s_g / P_s \\ P_{refl,b} &= P_{inc,b} s_b / P_s \end{aligned} \tag{3.6}$$

where P_{inc} is the power of the incident photon.

3.2.3 Photon storing

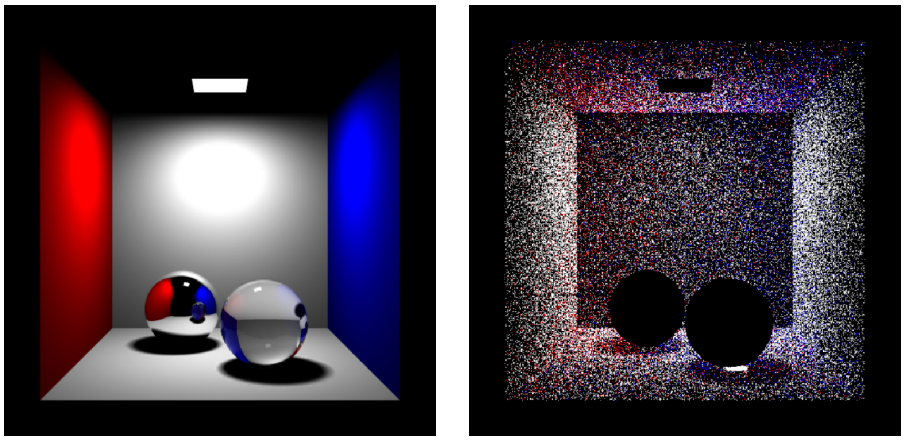
Photons are stored only when they hit diffuse surfaces. The reason is that hitting in specular surfaces don't give any useful information because the probability of having a matching incoming photon from the specular direction is almost zero. So, the best way to obtain accurate specular reflections is to trace a ray in the mirror direction using classic ray tracing. A photon can hit the geometry several times along its path, so it can be stored more than once (including when it is absorbed if the hit is on a diffuse surface). As we have mentioned, all the photon interactions are stored in a global data structure called *photon map*. In Figure 3.4, we can see a cornell-box scene rendered with traditional ray tracing and its corresponding photon map.

3.3 Photon Map Data Structure

The photon map, apart from naming the algorithm, is a representation of all the stored photons in the model. As we previously mentioned, the photon map is decoupled of the geometry which means that photons are stored in a separate structure without any association with the geometry.

3.3.1 The data structure

Photons are only generated during the photon tracing pass. Instead, during the render pass the photon map is a static structure used to estimate the illumination. These estimations are based on the nearest photons stored at any point. To the efficient working of the technique, it is needed a compact and fast, locating nearest neighbors in 3D space, structure.



(a) (b)

Figure 3.4: Cornell box with glass and chrome spheres: (a) traditional ray traced image (direct illumination, specular reflection and transmission), (b) the photons in the corresponding photon map. Figure obtained from [16].

Furthermore, the structure should be able to handle non-uniform distributions due to caustic effects and the fact that photons hit surfaces. A good candidate to handle all these requirements is a *balanced kd-tree*. We will explain more details of this representation in Section 3.3.3.

3.3.2 Photon representation

For each photon-surface interaction (hit), the photon map stores its position, incoming photon flux and incident direction. We need to store the position to have separated the photon map representation from the scene. Flux, or power, is the physical magnitude which allows to estimate the radiance. The incident direction allows to sample the BRDF during the rendering pass and, moreover, it allows to know which face (front or back) has been hit by the photon.

For efficiency reasons, the stored photons are divided into two photon maps:

Caustic photon map: contains photons that have done at least one specular reflection before hitting a diffuse surface: LS^+D.

Global photon map: an approximate representation of the global illumination solution for the scene for all diffuse surfaces: $L\{S \mid D \mid V\}^* D$.

where L represents the emission of the light, S represents a hit with a specular surface, D represents a interaction with a diffuse surface and V represents volume scattering.

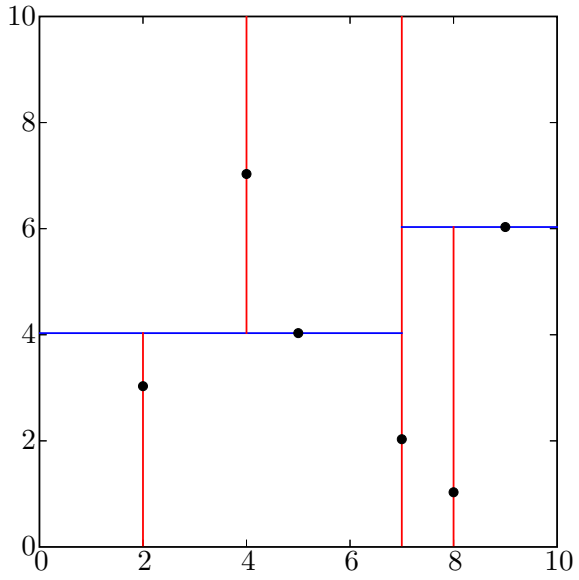
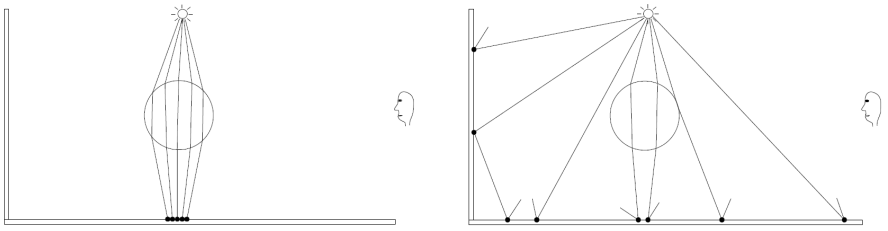


Figure 3.5: An example of a *kd-tree*.

The reason of two different photon maps is based on the idea that caustic photon map needs high quality and, therefore, more photons. For this reason, a separate photon tracing pass is performed for it. This will become clearer after reading section 3.5. In figure 3.6 it is illustrated the construction of the caustics photon map and global photon map in a scene with a glass ball and two diffuse walls.



(a) (b)

Figure 3.6: An example of building (a) the caustics photon map and (b) the global photon map. Figure obtained from [16].

3.3.3 The balanced kd-tree

A kd-tree is a space-partitioning structure for organizing points in k-dimensional space. See Figure 3.5. More specifically, it is a multi-dimensional binary search tree in which each node is used to partition one of the dimensions. In figure 3.5 an example of it is shown.

It is possible to locate one photon in the kd-tree with n photons in $O(\log n)$ time on average but in $O(n)$ worst time if the tree is very skewed. Although tracing photons randomly through a model can make one think that the tree built dynamically would be well balanced. The reality is that this does not happen. As the generation of the photons is based on the projection map and as the models often contain zones with the same direction of reflection, the kd-tree normally results in a skewed tree. Since the tree is build once and used many times and, given the fact that if the tree is balanced the worst time becomes $O(\log n)$, balancing the tree is a recommended option.

Balancing a kd-tree is like balancing a binary tree except the choice at each node of a splitting dimension. The choice of a splitting dimension is based on the distribution of points within the set. More specifically, it is chosen as the dimension which has the largest maximum distance between points. When it is chosen, the median of the points in that direction is chosen as root node and the left and the right trees are constructed from the two sets separated by this point.

3.3.4 Locating the nearest photons

Locating the nearest neighbor points is very important to the efficiency of the method. Fortunately, the kd-tree allows to use efficient search algorithms based on standard algorithms for binary trees. Jensen proposed an algorithm for locating the neighbors similar to range searching becuase the objective was to locate photons within a given volume.

The algorithm starts at the root of the kd-tree and add photons to a list if they are within a certain distance. The list is implemented as a priority queue because it is needed a efficient implementation to pop the furthest element when the list is full and a new closer element is found. Furthermore, the author proposed to use squared distances to avoid the root calculation per distance check achieving more efficiency.

In Algorithm 1, it is given the pseudocode of the search algorithm. Note that is necessary to provide an initial maximum search radius. This radius is important because a well-chosen number reduces the number of photons tested, while a radius too low will introduce noise in the estimation and a too high radius will compromise performance.

Algorithm 1 Pseudocode for locating the nearest photons in the photon map. Given a photon map, a position x and a max search distance d^2 this recursive function return a heap h with the nearest photons. The search is initiated with *locate_photons*(1). Algorithm from [14].

```
1: procedure LOCATE_PHOTONS(p)
2:     if 2p + 1 < number of photons then          ▷ Examine child nodes.
   Compute distance to plane (just a substract).
3:         δ = signed distance to splitting plane of node n
4:         if δ < 0 then      ▷ We are left of the plane - search left subtree
   first.
5:             locate_photons(2p)
6:             if δ² < d² then                          ▷ Check right subtree.
7:                 locate_photons(2p + 1)
8:             end if
9:         else       ▷ We are right of the plane - search right subtree first.
10:            locate_photons(2p + 1)
11:            if δ² < d² then                          ▷ Check left subtree.
12:                locate_photons(2p)
13:            end if
14:        end if
15:    end if
16:    δ² = squared distance from photon p to x   ▷ Compute true squared
   distance to photon
17:    if δ² < d² then              ▷ Check if the photon is close enought?
18:        insert photon into max heaph
19:        d² = squared distance to photon in root node of h       ▷ Adjust
   maximum distance to prune the search
20:    end if
21:    return h
22: end procedure
```

3.4 The Radiance Estimate

The photon map represents incoming flux (ϕ) in the model. Each photon transports a part of the light source power, so when a photon hits a region, it implies this region is receiving some illumination from a light source (directly or indirectly). In that way, the irradiance of a given region is estimated computing the photon density $d\phi/dA$.

In a similar way, to compute radiance estimates at any non-specular surface point in any given direction it is needed to integrate the incoming flux. This can be done using the expression for reflected radiance:

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) L_i(x,\vec{\omega}') \left|\vec{n}_x \cdot \vec{\omega}'\right| d\omega_i' \tag{3.7}$$

where L_r is the reflected radiance at x in direction $\vec{\omega}$. Ω_x is the hemisphere of incoming directions, f_R is the BRDF at x and L_i is the irradiance. Using the relationship between radiance and flux we can compute the irradiance:

$$L_i(x,\vec{\omega}') = \frac{d^2\phi_i(x,\vec{\omega}')}{\cos\theta_i d\omega_i' dA_i} \tag{3.8}$$

and the radiance estimation can be rewritten as

$$L_r(x,\vec{\omega}) = \int_{\Omega_x} f_r(x,\vec{\omega}',\vec{\omega}) \frac{d^2\phi_i(x,\vec{\omega}')}{dA_i} \tag{3.9}$$

The incoming flux ϕ_i is approximated locating the n photons with shortest distance to x in the photon map. Each photon p has the power $\Delta\phi_p(\vec{\omega}_p)$ and, in that way, the radiance estimation can be approximated as:

$$L_r(x,\vec{\omega}) \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega}) \frac{\Delta\phi_p(x,\vec{\omega}_p)}{\Delta A} \tag{3.10}$$

The equation still contains ΔA, which is related to the density of the photons around x. The author proposed to assume that the surface is locally flat around x and to compute this area by projecting the sphere that contains the n nearest photons to x onto the surface and to use the area of the resulting circle:

$$\Delta A = \pi r^2 \tag{3.11}$$

where r is the radius of the aforementioned sphere. So the equation for computing the reflected radiance at a surface using the photon map can be rewritten as:

$$L_r(x,\vec{\omega}) \approx \frac{1}{\pi r^2} \sum_{p=1}^{N} f_r(x,\vec{\omega}_p,\vec{\omega}) \Delta\phi_p(x,\vec{\omega}_p) \tag{3.12}$$

This estimate is an approximation, so the accuracy depends on the number of photons used in the photon map and in the formula. As a sphere is used to locate the photons, area estimation can be wrong in corners and shapes. However, the error can be reduced by increasing the number of photons and reducing the size of the kernel.

3.5 Visualizing the photon map

The photon map is view independent, so having the photon map computed and knowing how to estimate the radiance is all what is needed to render the final image. This final image is computed using ray tracing where each pixel is an average of some sample estimates. The radiance of each sample is computed tracing a ray from the observer through a pixel. Its value is the outgoing radiance in the direction of the ray in the point of intersection.

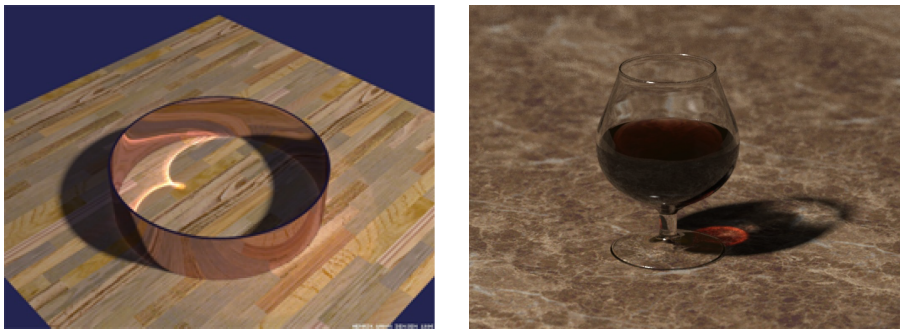


Figure 3.7: Caustic examples generated with photon mapping technique for a ring and a cognac glass. Left figure obtained from [14] and right figure from [16].

The outgoing radiance (L_o) can be described as:

$$L_o(x, \vec{\omega}) = L_e(x, \vec{\omega}) + L_r(x, \vec{\omega}) \tag{3.13}$$

where L_e is the emitted radiance and L_r is the reflected radiance that can be computed by integrating the contribution of the incoming radiance L_i as shown in Section 3.4.

$$L_r(x, \vec{\omega}) = \int_{\Omega_x} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}') \cos\theta_i d\omega_i' \tag{3.14}$$

To avoid using Monte Carlo integration techniques because of their cost, the author proposed to separate the BRDF into a sum of a specular component ($f_{r,s}$) and a diffuse component ($f_{r,d}$) such that:

$$f_r(x, \vec{\omega}', \vec{\omega}) = f_{r,s}(x, \vec{\omega}', \vec{\omega}) + f_{r,d}(x, \vec{\omega}', \vec{\omega}) \tag{3.15}$$

Furthermore, he proposed to classify the incoming radiance using three components:

- $L_{i,l}(x, \vec{\omega}')$ is direct illumination by light coming from the light sources.
- $L_{i,c}(x, \vec{\omega}')$ is caustics - indirect illumination from the light sources via specular reflection or transmission.
- $L_{i,d}(x, \vec{\omega}')$ is indirect illumination from the light sources which has been reflected diffusely at least once.

Then, the incoming radiance can be expressed as the sum of these three components:

$$L_i(x, \vec{\omega}') = L_{i,l}(x, \vec{\omega}') + L_{i,c}(x, \vec{\omega}') + L_{i,d}(x, \vec{\omega}') \tag{3.16}$$

Taking into account that changes, the incident radiance expression can be expresed as:

$$\begin{aligned} L_r(x, \vec{\omega}) = & \int_{\Omega_x} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}') \cos\theta_i d\omega_i' + \text{ (Direct illumination)} \\ & \int_{\Omega_x} f_{r,s}(x, \vec{\omega}', \vec{\omega})(L_{i,c}(x, \vec{\omega}') + L_{i,d}(x, \vec{\omega}')) \cos\theta_i d\omega_i' + \text{(Specular refl.)} \\ & \int_{\Omega_x} f_{r,d}(x, \vec{\omega}', \vec{\omega}) L_{i,c}(x, \vec{\omega}') \cos\theta_i d\omega_i' + \text{ (Caustics)} \\ & \int_{\Omega_x} f_{r,d}(x, \vec{\omega}', \vec{\omega}) L_{i,d}(x, \vec{\omega}') \cos\theta_i d\omega_i' \text{ (Multiple diffuse reflections)} \end{aligned} \tag{3.17}$$

(a) (b) (c)

Figure 3.8: Other phenomena can be simulated adapting the technique: (a) participating media, (b) subsurface scattering and (c) diffraction. Figure obtained from [16].

This is the equation used in photon mapping to compute the reflected radiance in a surface. Jensen distinguished two situations: one accurate

and the other approximate. The accurate computation is done when the surface is seen directly by the observer or via a few specular reflections. The approximate computation is done if the ray intersecting the surface has been reflected diffusely since it left the observer. Here is the reason of having two photon maps and they are used as follows:

- Direct illumination for visible surfaces is computed using Ray Tracing.
- Specular reflections are computed using Ray Tracing.
- Caustics are computed using caustics photon map.
- Multiple diffuse reflections are computed using global photon map.

The photon mapping technique can be adapted to simulate more complex phenomenas. For example, the Jensen proposed some modifications to simulate participating media, subsurface scattering or diffraction phenomena. An example of these phenomenas can be seen in figure 3.8.

Chapter 4

Our Approach

4.1 Overview

Based on our conclusions on the current state of the art in photon mapping and urban rendering and, after analyzing the problems of our scenario, we have decided to use a hybrid technique, where near street lights are simulated with photon mapping and far street lights are simulated using an approximation based on direct *OpenGL* rendering. Figure 4.1 shows a diagram of the different steps of the propossed technique.

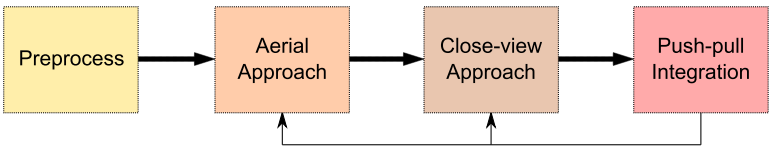


Figure 4.1: Overview of the technique.

When the observer is far away, we use a precomputed global approximation for the whole city. The approximation consists of rendering a textured quad centered in the position of each street light to generate an approximate light map for the whole city. The information of the light map is used to compute the illumination of both the floor and the facades. Although it is a rough approximation, the quality is good enough for our purpose because it will be only used for far away visualizations where the details are not appreciable.

Instead, when the observer is near a street light, its contribution to the illumination is computed more accurately using photon mapping. For this computation we use a variation of the technique which splats the energy instead of gathering it. We store the hit of the photons in two photon maps: one for the floor and another for the walls. In that way, we achieve accurate and physically realistic results when the visualization requires this kind of details.

Since we have two totally independent solutions, we propose a technique to integrate both based on a method inspired from classic hierarchical radiosity, namely the push-pull methods. In this method, we use information of the light map to compute the photon map and, once it is computed, we reuse this information to improve the light map with more realistic information.

4.2 Model preprocessing

Before providing details about the illumination computations, we should explain all the preprocessing needed for the proper working of the method. Furthermore, we introduce the urban's model visualization method followed. Figure 4.2 shows this step in the overview of the technique.

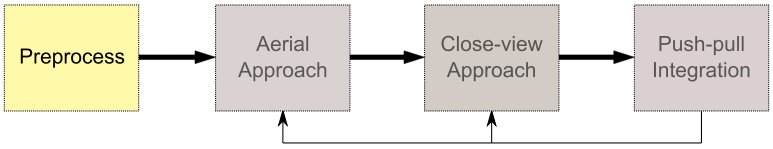


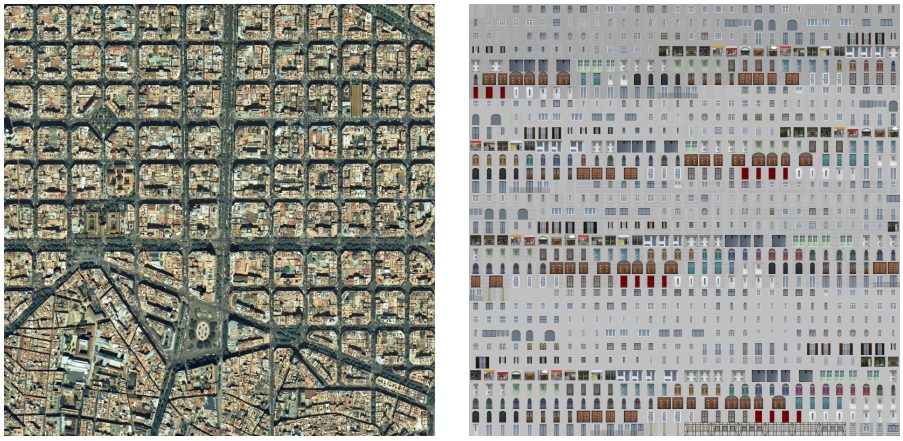
Figure 4.2: Section 4.2 represents the first box of the diagram.

4.2.1 Urban rendering

Urban rendering is a very studied area in computer graphics as we have mentioned in Chapter 2. In this thesis we have followed the method proposed by Argudo et al. [3] and we have used the same model of the Barcelona city. See figure 4.3a.

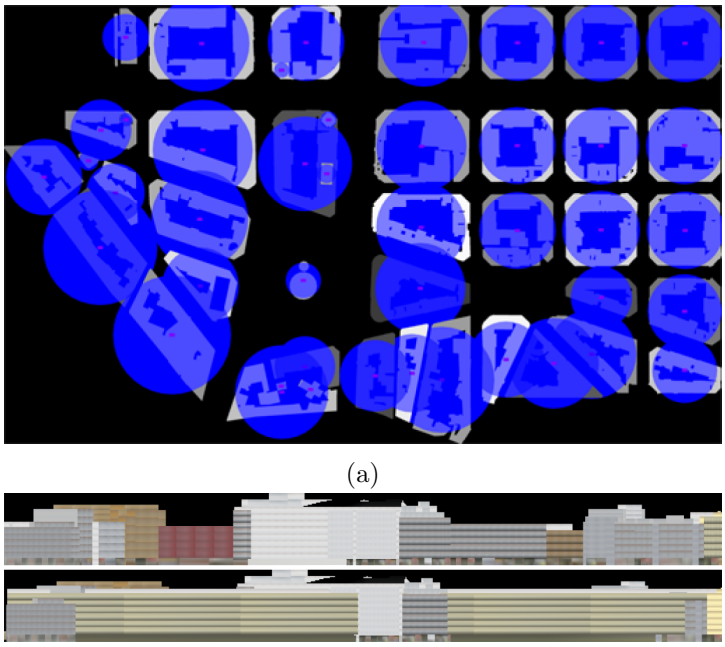
The model consists of a set of textured polygonal meshes that represent the buildings and the floor. Figure 4.3 shows an example of these textures. The texture of the floor is a picture of an orthophoto of the city with a resolution of 4096x4096. Furthermore, they construct a height map encoding the elevation of each point of the building as seen from the top. On the other hand, the facades are stored in a texture array of texture maps with a resolution of 1024x1024. These textures encode the RGB color and a flag indicating if the surface is specular or diffuse.

Moreover, we have based the structure of our facade photon map on the solution proposed by Argudo et al. [3]. They propose a cylindrical parametrization because the block shape is usually square or rectangular. Furthermore, they generate two textures for the building exterior and interior facades with the same parametrization. Thus, we have reused their facade parametrizations and textures generated for that purpose. In Figure 4.4a, we can see how the approximation is good enough and an example of the texture generated is shown in Figure 4.4b. The cylindrical coordinates of the parametrization are computed as:



(a) (b)

Figure 4.3: Example of textures used in the model. Figure (a) corresponds to the orthophoto used in the model, and figure (b) corresponds to one of the textures used with the facades.



(a)

(b)

Figure 4.4: In (a) is shown how the parametrization is good enough for its purpose. In (b) an example of the texture generated with this parametrization is shown. Figure from [3].

$$s = \frac{\arctan x/z + \pi}{2\pi}$$
$$t = \frac{y - y_{min}}{y_{max} - y_{min}} \tag{4.1}$$

It is important to notice that the orthophoto texture is used for the floor of the city and for the roof of the buildings. For this reason, from now on, since now every time that we mention a technique, algorithm, formula or whatever for the floor of the city, it is also used for the roof of the buildings.

4.2.2 Light positions

One important thing before being able to simulate night illumination is to know where the lights are placed. As we did not have this information, we developed a simple method to place street lights. After an inspection through the streets of Barcelona to find out how the street lights are situated in real life, we observed that, usually, they are approximately placed at twenty meters from one another, two or three meter away from buildings and about ten meters high. The method that we propose for placing lights consists of rendering an initial image with building silhouettes and use image segmentation operations to place the street lights. This image-space approach is robust against model degeneracies such as self-intersections and non-watertight geometry. This process can be divided in the following steps:

1. **Rendering:** First of all, we render the initial image. As we have information of the building heights, we render the scene from top with an orthogonal projection. The dimensions of the viewport used are chosen preserving the aspect ratio that the zone rendered has in real life, and such that each pixel corresponds to 2x2 meters. During that process, we render all the buildings in black over a white background. Result shown in Figure 4.5.

2. **Selecting the surrounding region:** Once this image has been generated, we apply morphological operations to detect all floor pixels within four meters from the building facades. First, all the pixels corresponding to the floor are selected. For this, we apply a region growing operation seeding with white pixels until no white pixel remains unselected. Then we apply twice a erosion operation with $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$ as operator. Once it is done, we invert the selection to have the buildings and their surrounding region selected. After that, we unselect the buildings by computing the difference between the actual selection minus the buildings selection. To compute the selection of the buildings, we apply a region growing operation seeding with black pixels until no

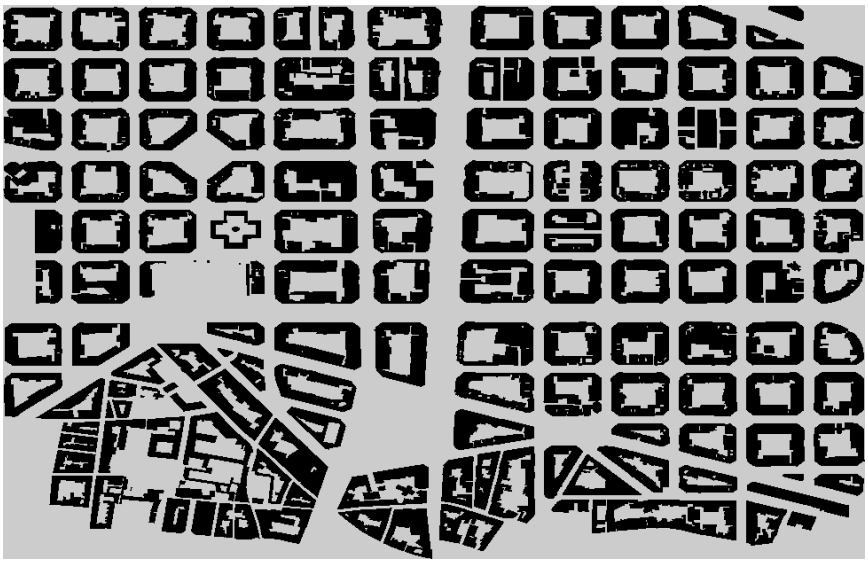


Figure 4.5: Result of the rendering step in the street light allocation process.

black pixel remains unselected. When we apply this difference operation, we mark the selected area by painting these pixels in red. Result shown in Figure 4.6.

3. **Placing lights:** After selecting and marking a perimeter of approximately four meters around the buildings, we proceed to place the street lights. For this, we iterate over all the selected pixels and we place a street light on the position of this pixel if it satisfies the following conditions: (Result shown in Figure 4.7)

    - At least one of the surrounding pixels is a white pixel.
    - There is no other street light inside a square of 20x20 pixels centered at this pixel.

With this method we place the street lights on the city guaranteeing that they will be three meters away from the buildings and that every pair of lights will be distanced at least 20 meters. The height of the positions is 10 meters. As we have mentioned before, this pattern approximately matches that found in Barcelona city.

4.2.3 Histogram matching

Before starting to compute the light simulation, we must solve another problem. The orthophotos that can be obtained for any city are, in general, taken during the day. If we illuminate the city with this orthophoto, the result is not going to be natural: the city is going to have daily colors illuminated by

Figure 4.6: Result of the selecting surrounding region step in the street light allocation process.

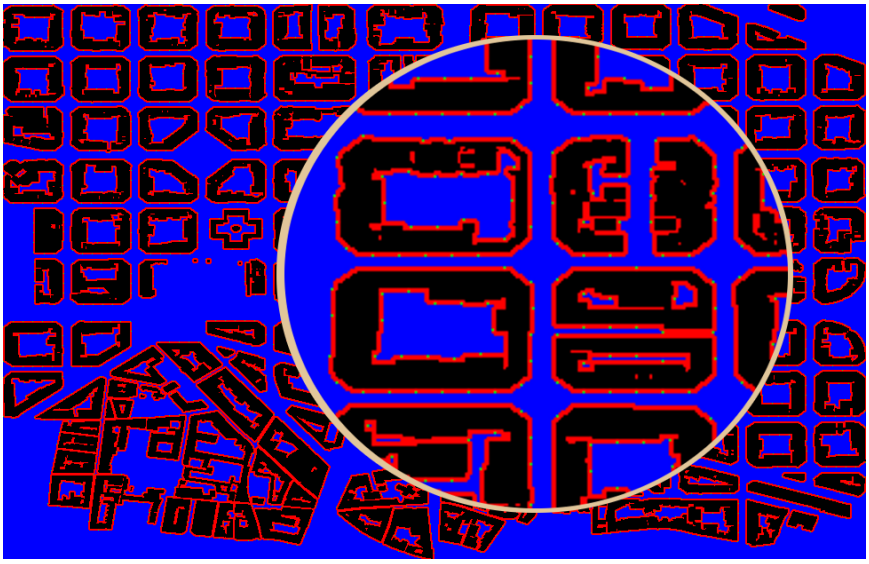


Figure 4.7: Result of the placing lights step in the street light allocation process. Background is blue to enhance the light positions.

street lights at night. Normally, the street lamps are high pressure sodium light which emit orange light: although there are exceptions, this kind of lamps is very common in most of the cities in America [4]. One possible solution to this problem could have been to emit orange light from our sources but, we decided to test another technique named *Histogram matching* [21] that allows us to continue emitting white light and to have a straight-forward implementation.

The purpose of this technique is, given a source and a target images, to modify the source image to its histogram matches the target image histogram. The authors describe that the method starts computing the histogram of the source and target images. They define the computation of the histogram H with B bins of width V of an image I as follows:

$$H = \{(h(1), v(1)), \ldots, (h(B), v(B))\}, \tag{4.2}$$

$$B = \left\lceil \frac{\max(I) - \min(I)}{V} \right\rceil, \tag{4.3}$$

$$h(i) = \sum_{p=1}^{N} P(I(p), i), \quad i \in [1, B], \tag{4.4}$$

$$v(i) = \min(I) + (i-1)V, \tag{4.5}$$

$$P(I(p), i) = \begin{cases} 1 & i = \left\lfloor \frac{I(p) - min(I)}{V} + 1 \right\rfloor, \\ 0 & \text{otherwise,} \end{cases} \tag{4.6}$$

where H is the set of all pairs $(h(i), v(i))$ for all $i \in [1, B]$ corresponding to the number of elements and value of the ith bin of the histogram. $I(p)$ is the value of the pth pixel of image I which contains a total of N pixels and $P(I(p), i)$ represents the probability of a pixel $I(p)$ belonging to a bin i. After that, the cumulative histograms of the source and target are computed as:

$$C_s(i) = \sum_{i=1}^{B} h_s(i) \tag{4.7}$$

$$C_t(i) = \sum_{i=1}^{B} h_t(i) \tag{4.8}$$

Finally, the resulting image I_r is computed matching source to target according to these two cumulative histograms:

$$I_r(p) = v_t\left(C_t^{-1}\left(C_s\left(\frac{I(p) - min(I) + 1}{V}\right)\right)\right) \tag{4.9}$$

Figure 4.8 shows the results of how this technique works with the orthophoto. We apply the technique to the three color channels (RGB) independently and the alpha channel remains equal to the source image because there is information encoded not related with color. As the resulting image was too dark, we introduced a small variation in the histogram matching technique. When we compute the histogram H of the target image, we force that for every channel:

$$I(p) = \begin{cases} 0 & I_R(p) < 25 \wedge I_G(p) < 25 \wedge I_B(p) < 25 \\ I(p) & \text{otherwise,} \end{cases} \tag{4.10}$$

Figure 4.8 also shows the result of this variation. In that way, without taking into account the darkest pixels, the method gives a better result for our purpose. The resulting image looks similar to the target image, but brighter than the original result. Moreover, although the histogram is not as similar as original result it preserves the main properties of the target histogram.

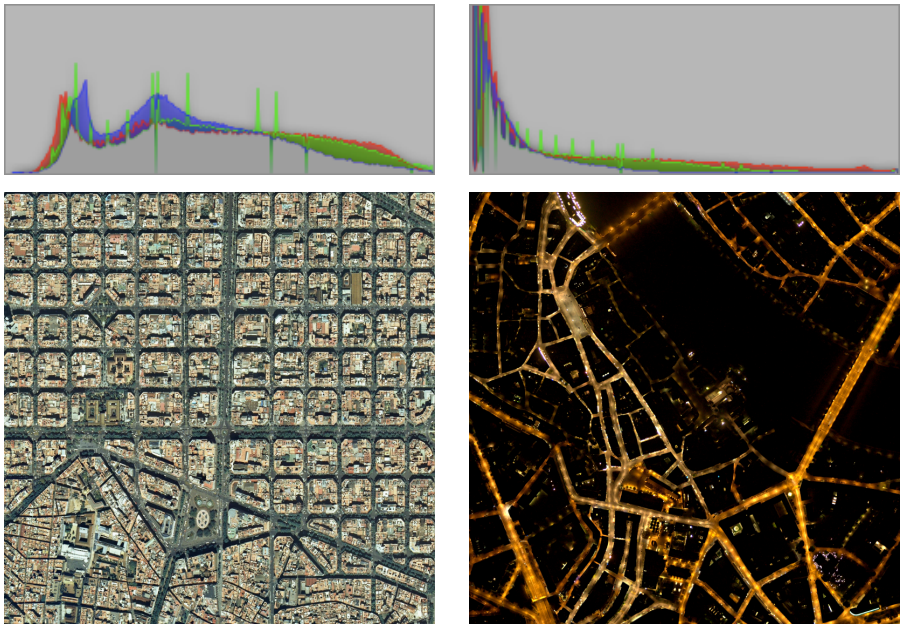
If we only apply this technique to the orthophoto texture, the contrast between floor and facades would have been too high. For this reason, this technique is also applied to all the textures corresponding to facades.

4.3 Aerial approach

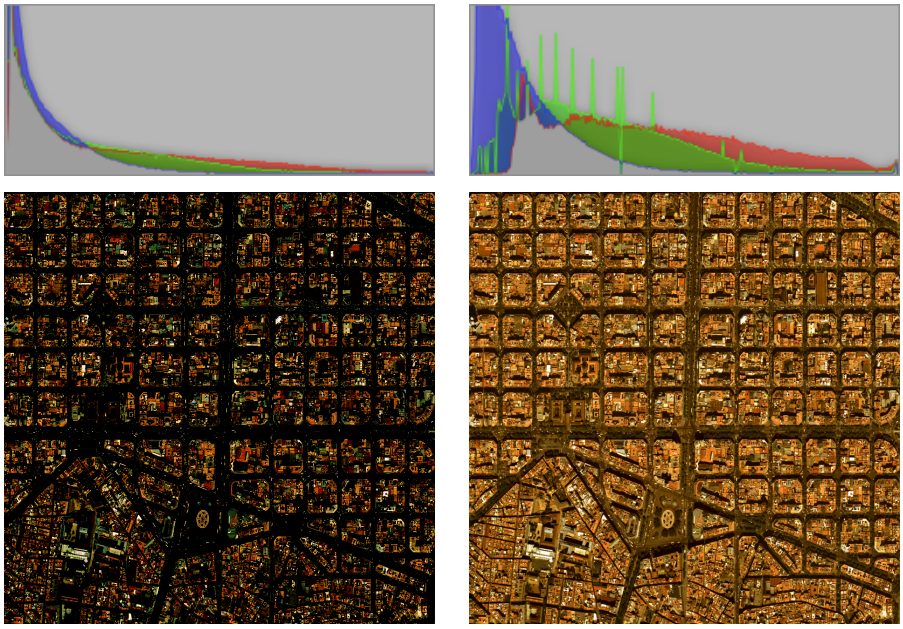
The first part of our technique consists on simulating the nocturnal illumination of the city for distant views. The high number of light sources in a city implies that rendering the whole city with physically realistic methods is impossible with current technologies. Instead, for distant views we can realize that the most important aspect is the computation of an approximation for the whole city with enough quality and a small render-time cost. The quality should be enough as long as the user can not appreciate the possible errors of the computation. We propose to generate a light map of the simulated illumination on the floor and use this information to illuminate both the floor and the facades. Figure 4.9 shows this step in the overview of the technique.

4.3.1 Light approximation

First, we should simulate the illumination of a street lamp. After trying cubic and gaussian functions without much success, we propose to use a



(a) Source (b) Target



(c) Original result (d) Our result

Figure 4.8: (a) Source image, (b) target image, (c) original histogram matching result image and (d) our result image (after the proposed variation) of the process applied to the orthophoto used in the model. In the first and third row, the corresponding histogram is shown for every image and, in the second and fourth row, the images are shown.

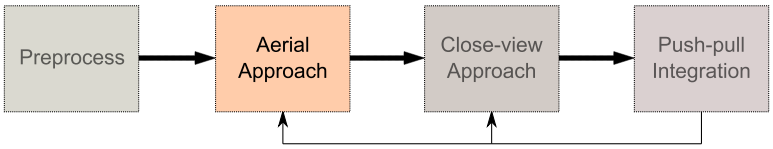


Figure 4.9: Section 4.3 represents the second box of the diagram.

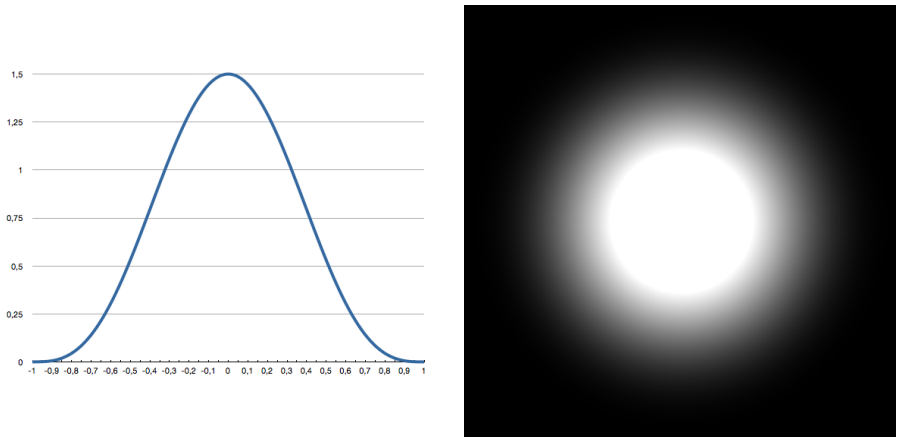


Figure 4.10: Left figure shows the curve followed in the approximation of the light of a street lamp and right figure shows the result of this light simulation.

phong-like formula:

$$\begin{aligned} I(x,y) &= A\cos^n \frac{r(x,y)}{r_{max}} && \text{if } r(x,y) \leq r_{max} \\ I(x,y) &= 0 && \text{if } r > r_{max} \end{aligned} \tag{4.11}$$

where A and n are two user-defined parameters which define the amplitude and the width of the curve, $r(x,y)$ is the distance to the light position and r_{max} is the radius of the influence area of the light.

This formula represents an adaptive way to approximate illumination based on the Phong formula. It allows a smooth transition between the most powerful part of the illumination and the dimmest one. In our implementation we have defined the influence radius (r_{max}) of 30 meters because it allows the interaction between street lights as they are located twenty meters far away from each other. Furthermore, in Section 4.4.3 there are given more arguments for this. In Figure 4.10, it is shown one example of this curve and of the light simulated with it.

4.3.2 Light map generation

Once the approximation algorithm has been decided, we should use it to generate the light map. The light map is a single-channel texture where we store the illumination (monochromatic) energy of the approximation for every point of the floor of the scene. This texture is built by painting the influence area of every street light with the formula 4.11.

Generating the light map consists of drawing a textured quad at every street light position taking advantage of the OpenGL pipeline. We have built a *Vertex Buffer Object* (VBO) with the street light positions as vertices, obtained in Section 4.2.2. To generate the texture we have rendered this VBO with an orthogonal projection adjusting the viewport to the floor of the scene. For every vertex rendered, we generate a quad of size $2r_{max}$x$2r_{max}$ in the geometry shader. Each quad generated is textured as we have mentioned before. Enabling blending during the render allows to express the influence between street lights in the light map. The information is stored as float value in a float texture.

This method is a fast way to generate the light map. Furthermore, roughly speaking, as the texture is generated once and used always, this reduces even more the importance of the time spent generating it. See Section 4.5. In Figure 4.11, it is shown an example of a light map generated for our scene.

4.3.3 Visualizing the aerial approximation

Finally, as we have mentioned before, the information of the light map is used to compute the illumination of the scene during the render step when the observer is far away from the city. The idea can be divided in two parts: floor illumination and facade illumination.

Computing the illumination of the floor is very simple. For every point of the floor we compute its illumination querying its corresponding texel in the light map and using its value. For the facades, the algorithm works in a more approximate way. For every point of the facade, its illumination corresponds to the illumination of the floor in this point in the light map dimmed by a factor depending of the height of the point and the height of the street lights. This is computed as follows:

$$L_i' = \max(0, (height - y)/height) * L_i; \tag{4.12}$$

where L_i' is the irradiance of the facade's point, L_i is the irradiance of the projection of the facade's point in the light map, $height$ is the height of the street light and y is the height of the facade's point.

As it happens in the real world, a point of the city is never completely dark, we add to the illumination computation of a pixel an ambient component that gives a very low light to every point. This algorithm is used in the *fragment shader* as it is described in the Algorithm 2.

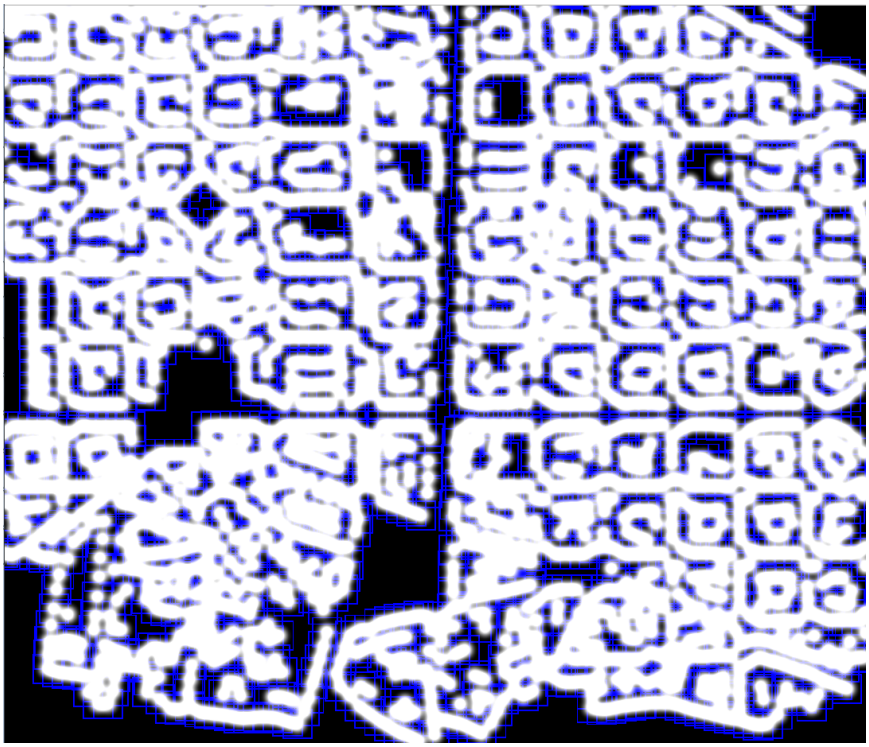


Figure 4.11: This figure shows an example of the light map of the aerial approach. The blue quads shown in the figure represent the size of the quads rendered during its construction. Although they are not a part of the light map, showing them can help to understand its construction.

Algorithm 2 Fragment shader's pseudocode for rendering the scene for distant views following the aerial approach and using the light map.

```
1: procedure RENDER(Pixel p)
2:     if p ∈ Floor then
3:         color = texture2D(orthophoto, textureCoordinates(p))
4:         irradiance = texture2D(lightMap, textureCoordinates(p))
5:     else if p ∈ Facade then
6:         color = texture2D(facadesTexture, textureCoordinates(p))
7:         pFloor = GetFloorPosition(p)
8:         irradiance = texture2D(lightMap, textureCoordinates(pFloor))
9:         irradiance = max(0, (StreetLightHeight − getHeight(p))/StreetLightHeight) ∗ irradiance
10:    end if
11:    fragmentColor = color ∗ (ambientComponent + irradiance)
12:    return fragmentColor
13: end procedure
```

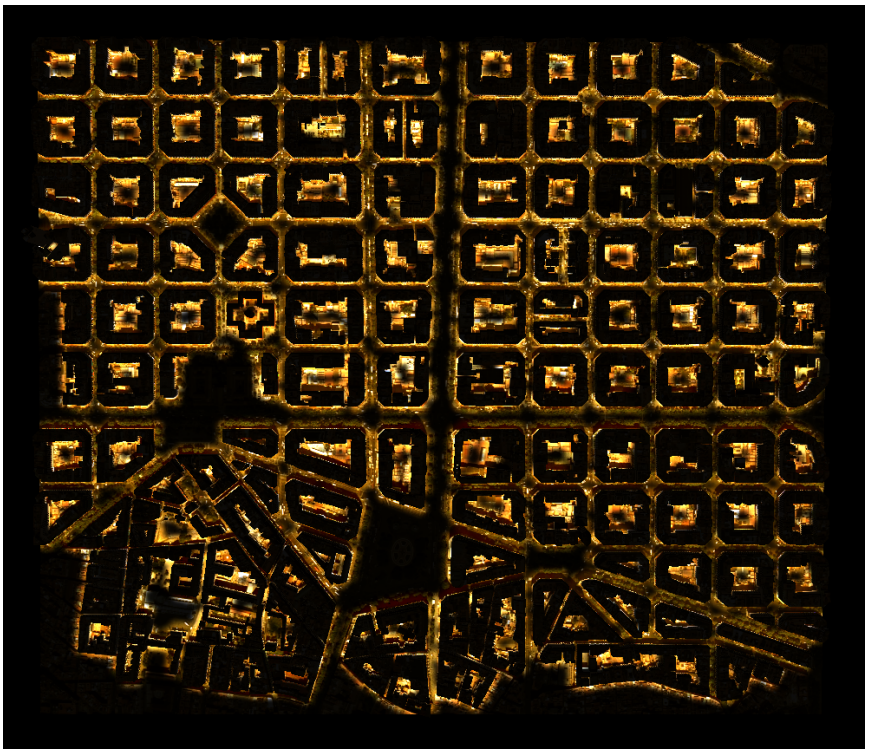


Figure 4.12: Rendering of the visualization approach for a distant view.

The primary goal of this approach is speed. The generation step consists only of a render pass done once. The visualization of this approach it is also not expensive. Only one extra texture access is needed for every pixel. As we can see in the Figure 4.12, the quality of the approximation is good enough for distant views fulfilling our objectives.

4.4 Close-view approach

The second part of our technique consists of simulating the nocturnal illumination of the city for close views. Our objective is to compute the illumination of the whole city as aerial views do but, in this case, it is very important to make the simulation as physically realistic as possible because the user would be close enough to appreciate the details of the illumination. In addition, as important as the quality, it is the time cost of the algorithm. It should be fast enough to allow real-time renderings. These are the reasons why we use photon mapping for this approach, a method which allows realistic global illumination computations in real time [20]. But even for a real time technique it is impossible to compute the illumination of the whole city because the number of lights is too high. For this reason we will only

compute a group of lights which are close enough to the observer. For all the lights computed with this approach we use the same technique, so we introduce the technique for a unique light and in Section 4.4.6 we introduce how the structures are modified when there are more than one street light. Figure 4.9 shows this step in the overview of the technique.

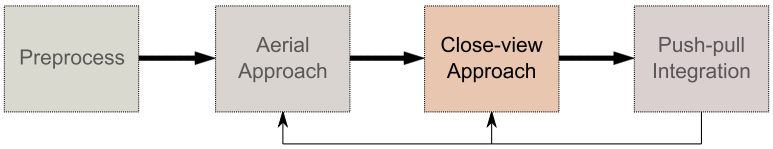


Figure 4.13: Section 4.4 represents the third box of the diagram.

4.4.1 Structuring street lights

As we mentioned before, we propose a method to select the group of lights that are going to be computed every moment. The query should be fast and the number of lights to compute every frame should be low or it will be impossible to achieve real-time rates.

One structure to divide the space in a fast and simple is a regular grid. The grid consists of subdividing the space with equal-size cells. The construction is trivial and the access is very fast. Although there are other structures that can better manage space (e.g. a quad-tree), for our simple purpose the grid is enough.

The grid has been built with a cell size of 40x40 meters. This cell size means, in our scenario, that the maximum number of lights per cell is on average five. After studying it, we realized that five lights per cell is an excellent tradeoff. A larger cell size could increase the computing time of the photon mapping approach, losing the real-time condition. In Figure 4.14 it is shown how the grid covers the scene and how the lights are distributed on the grid.

Once the space has been divided by the grid and each element of the grid contains a set of street lights, it is necessary to define how this information is going to be used to know what lights must be computed with photon mapping. The idea is very simple, at each position of the camera it would be checked if any street light must be computed. As the photon mapping will only be working for near views, the grid will be used when the height of the observer position is lower than user defined threshold $height_{min}$. If it happens, the grid will be queried to know what street lights are in the grid cell that corresponds to the projected camera position. Then the photon mapping approach will be computed for each of the lights of this set.

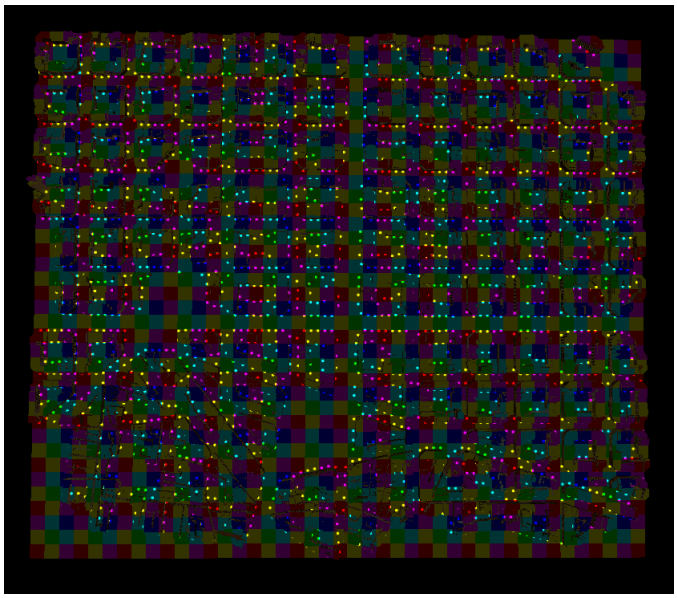


Figure 4.14: It can be shown how the scene is subdivided with the grid structure and how the lights are distributed over the subdivision.

4.4.2 Scope

Photon mapping is the technique chosen simulate the physically realistic illumination for close views. Our implementation introduces important differences respect classic photon mapping. One of these important differences is the photon map structure. In the original technique they generate a photon map for the diffuse reflections on the whole scene and a photon map for the caustics. In our case, due to the structure of our scene we use a photon map for the floor and a photon map for the facades, and they only cover the region of interest instead of the whole scene. This allows us to use higher resolutions in the photon map and, therefore, higher quality in our computations. This separation is the reason for storing the surface type hit by the photon as we propose in the Section 4.4.3. (I.e., floor, interior facade or exterior facade).

The floor photon map covers a squared region centered on the street light position. The size of the square has been chosen after the study showed in Table 4.1. We choose the same size for this region than for the quads used in the aerial approach introduced in Section 4.3.1. The size chosen is 60 meters, so $2r_{max} = 60$ meters in the aerial approach notation. This choice is backed up by the fact that it allows us a high resolution of the floor photon map, discarding only the 2.13% of the photons traced and, therefore, maintining the high quality of the approximation.

The facades photon map actually consists of two photon maps: one for exterior facades and another one for interior ones. This subdivisions

Square size	% Photons hit
30 meters	79.49%
60 meters	97.87%
90 meters	99.73%
120 meters	99.99%

Table 4.1: Table that shows the percentatge of the photons traced for a typical street light that hit in the floor inside of a squared region of different sizes.

allows to have separated the radiance estimation for the different parts of the building to have a quite simple an elegant solution. As they work in the same way we will only present how it works for general facades photon map and it can be extrapolated to both photon maps. The facade photon map is a texture atlas where each element of the atlas represents the photon map of the facades of one bock. This block photon map is filled using the cylindrical parametrization proposed by Argudo et al. [3] and mentioned in Section 4.2.1.

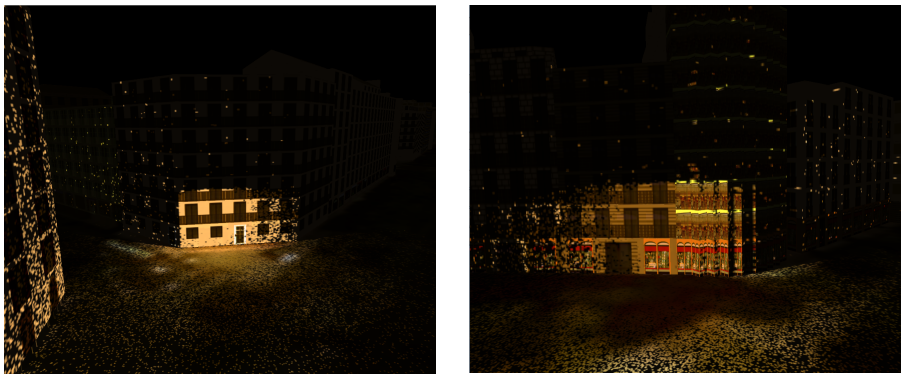


Figure 4.15: This image shows the photons hitting the geometry. All the photons have been thrown with the same energy and, during the tracing process, they don't sample the BRDF.

4.4.3 Tracing the photons

The first step to compute the illumination of a street light is to emit the photons and to trace them through the scene. To trace the photons we have used the *NVIDIA Optix* ray-tracing engine which exploits the parallelism of the graphics card. During the computation, we emit NxN photons from the light source. They are thrown in random directions $\vec{\omega_p}$ chosen by sampling a disk and projecting them up to a unit hemisphere. Then the directions are transformed to light space using an ortonormal basis so that the normal of

the plane that crosses the hemisphere is the street light direction. The rays bounce until they terminate at a diffuse surface or the number of bounces is greater than a user-defined limit $depth_{max}$ to avoid infinite reflections between specular surfaces.

The initial energy should represent the characteristics of the light. For this reason, the initial energy of each photon p is set related to its direction so that it has the same energy that it was proposed in the aerial approach (See Formula 4.11).

The behavior of the method is different according to the material of the surface hit x. When the photon hits a diffuse surface, we store the hit in the photon map. More specifically, we store the position of the bounce x, the irradiance energy $L_{i,p}$ and the kind of the element hit (floor, front facade or back facade). After that, we compute the direction of the bounce $\vec{\omega_p}$ as when it is emitted from the source, but using the normal of the surface instead of the street light direction. In addition, the radiance energy is computed sampling the BRDF at the hit point $f_r(x, \vec{\omega_p}', \vec{\omega_p})$ as follows:

$$L_{r,p,d}(x, \vec{\omega_p}) = f_r(x, \vec{\omega_p}', \vec{\omega_p}) L_{i,p}(x, \vec{\omega_p}')$$

On the other hand, when the photon hits a specular surface, the ray is reflected using the normal of the surface. The radiance energy is computed using the Fresnel coefficient kS as follows:

$$L_{r,p,s} = kS L_{i,p}(x, \vec{\omega_p}') \tag{4.13}$$

Although generating a unique ray at every hit of the photon can seem an error, this is caused because it is an approximation depending on the initial number of rays emitted from the light source N^2. We can observe that this computation for every hit is equivalent to the radiance estimation formula in classic photon mapping, but adjusting the number of photons p to $n = 1$:

$$L_r(x, \vec{\omega}) \approx \sum_{p=1}^{n} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}'). \tag{4.14}$$

But, if $n = \infty$, it is equivalent to the radiance estimation formula:

$$L_r(x, \vec{\omega}) = \int_{\Omega_x} f_r(x, \vec{\omega}', \vec{\omega}) L_i(x, \vec{\omega}'). \tag{4.15}$$

This means that, when infinite photons are thrown, at each bounce infinite rays are generated and then the proposed solution converges to the correct one.

In Figure 4.15, we can see an example of how the photons hit the scene. In this figure, the photons are emitted with the same energy and without applying the BRDF of the surface. Thus, the reader can appreciate how the photons hit the geometry.

4.4.4 Generating the photon map

After the photons have been traced through the scene, the next step is to store the information of the hits in the photon map. As we have said before, we use a variation of photon mapping that consists on splatting the photons in the photon map. Splatting the photons means that, instead of storing a single point for every hit in the photon map, we extend the photon hit to the region that is influenced by this photon. Instead of storing the energy $\Delta\phi_p(x,\vec{\omega})$, we store the energy density at each point of the region as $\frac{\Delta\phi_p(x,\vec{\omega})}{\Delta A}$. Thus, we avoid the gathering step and, when we do the rendering, we can estimate the radiance of each point by directly accessing the photon map, which gives a result equivalent to that of classic photon mapping:

$$L_r(x,\vec{\omega}) \approx \sum_{p=1}^{n} f_r(x,\vec{\omega}_p,\vec{\omega})\frac{\Delta\phi_p(x,\vec{\omega}_p)}{\Delta A} \tag{4.16}$$

where A in our case is the area of the splat drawn instead of the area used to sample the radiance.

Splatting in both photon maps is done so that the size of the photon influence region is the same in world coordinates in both cases. If it was not like this, the difference between splatting sizes would introduce errors in the radiance estimation during rendering because, for two identical photons, their radiance would be different if they hit in the floor or in a facade.

In Figure 4.16 it is shown how the photons maps are built. In the left picture, the photons stored in the floor photon map after splatting the hits produced in the last photon tracing step. In the right image, it is shown the atlas of the exterior facade photon map after several computations. Each row of the atlas represents one of the blocks. The interior facade photon map is similar, and for this reason it is not shown.

4.4.5 Visualizing the photon map

The last step of this approach is to use the photon map to compute the illumination. The method consists of checking, for every pixel, if it is in the photon map and, if this happens, use the information stored to compute its illumination. As we have mentioned before, the rendering step is quite simple because in the photon map we store the density of the energy instead of the energy directly.

The Algorithm 3 represents a pseudocode of the fragment shader which renders the scene. We can see how this method is very similar for floor and facades. The unique difference is the textures used to compute the original color and the irradiance.

One of the objectives of this approach is to be fast. The generation step consists of tracing the photons and splatting them in the photon maps. This generation step is thought to allow real-time computations and, moreover, it

Algorithm 3 Fragment shader's pseudocode for rendering the scene for near views following the photon mapping approach and using the photon maps.

```
1: procedure RENDER(Pixel p)
2:     if p ∈ Floor then
3:         color = texture2D(orthophoto, textureCoordinates(p))
4:         if p ∈ FloorPhotonMap then
5:             irradiance = texture2D(floorPM, textureCoordinates(p))
6:         else
7:             irradiance = 0
8:         end if
9:     else if p ∈ Facade then
10:        color = texture2D(facadesTexture, textureCoordinates(p))
11:        building = getBuilding(p)
12:        if p is a FrontFacade point then
13:            if p ∈ FrontFacadePhotonMap then
14:                texCoord = getCoordinatesInAtlas(p)
15:                irradiance = texture2D(frontFacadePM, texCoord)
16:            else
17:                irradiance = 0
18:            end if
19:        else if p is a BackFacade point then
20:            if p ∈ BackFacadePhotonMap then
21:                texCoord = getCoordinatesInAtlas(p)
22:                irradiance = texture2D(backFacadePM, texCoord)
23:            else
24:                irradiance = 0
25:            end if
26:        end if
27:    end if
28:    fragmentColor = color * (ambientComponent + irradiance)
29:    return fragmentColor
30: end procedure
```

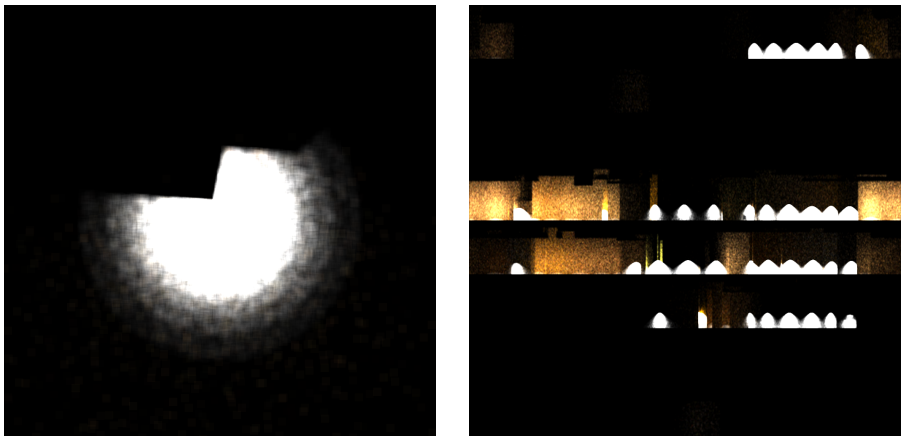


Figure 4.16: In this figure, photons maps are shown after several photon tracing steps. In left image we can see the floor photon map and in the right image the facade photon map.

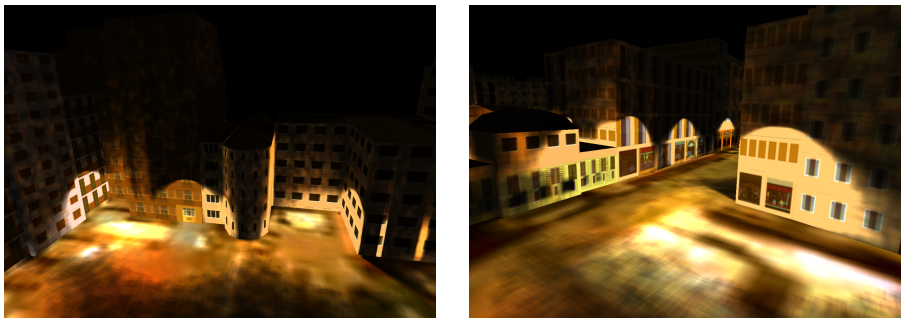


Figure 4.17: Renderings of the photon mapping approach for a near views.

is generated once and it can be used while the street light is being visualized. The visualization is also inexpensive. Only one extra texture access is needed for every pixel and some quick checks to know if the pixel is a part of the region of the photon map. As we can see in Figure 4.17, despite the poor quality of the texture maps in our test dataset, the quality of the photon mapping approach is very high and it is physically realistic.

4.4.6 Light cache

Until now, the method has been proposed to handle only a light. But as we mentioned in Section 4.4.1, the method should compute the illumination of a set of lights. The way to extend the photon mapping approach to manage several lights, is to adapt photon map structures to this purpose. In Section 4.4.4, photon mapping was introduced to store the floor hits of a unique light. Moreover, facade photon mapping was introduced as an atlas of block facades where the photon hits were stored. In this section we are going to

introduce the solution used.

The user can follow arbitrary paths during the inspection of the scene, so that it is impossible to predict what street lights may be computed with photon mapping approach each moment. Furthermore, if we try to store a photon map for each street light we would consume too much memory resources, or the resolution of the photon map would be too low for photo-realistic rendering. For this reason, we need a structure that allow us to know what street light must be computed and where they have to be stored in a very fast way.

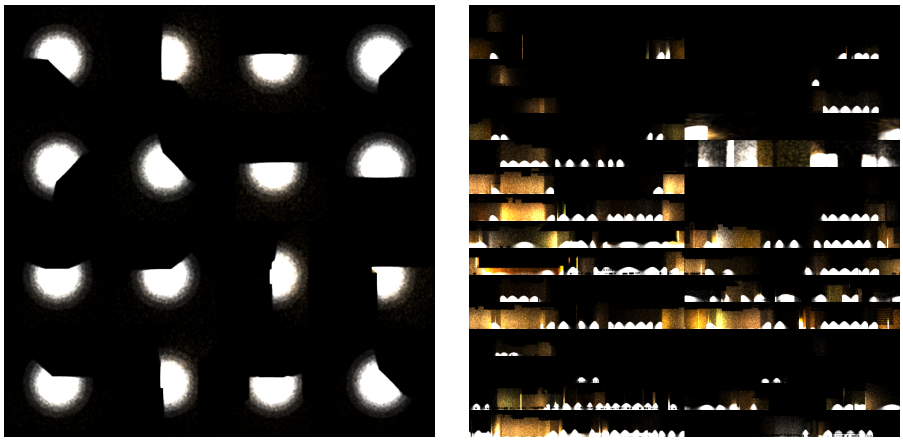


Figure 4.18: Example of the photon map atlases (left floor photon map atlas and right front facades photon map atlas) after a few navigation steps.

The solution proposed is to build atlases with the photon maps and use them as a cache with a *Last Recently Used* (LRU) policy. Although both the floor photon map and facades photon map are managed as a LRU cache, the way they are filled is a bit different. We use the grid structure to determine the street lights that contribute most to the current frame. For each light L, we check if L is in the floor photon map cache. If it is allocated in the cache, no more computation is needed because the floor photon map and the facade photon map already have the needed information, because the light has been previously computed. On the other hand, if it is not allocated and there are empty spaces, the cache will provide one of the empty slots. If there is not free slot, the last recently used position will be freed and its position will be assigned to this light. After that, the tracing step of the photon mapping approach is performed. Here, the floor photon map cache indicates the position of the floor photon map atlas of the current street light.

For the facades photon map, its cache is used during the photon tracing step. For each hit, it is checked what kind of element it hits. Remember that, if the photon hits the floor, it is already defined the atlas position

where this photon should be splatted. On the other hand, if the photon hits a facade, the facades photon map cache will be queried if the building hit is allocated in the cache. If it is allocated, the cache will indicate the position of the facade photon map atlas where the current photon should be splatted. If it is not allocated, the cache will look for an empty space. If there is not any empty space, the cache will free the last recently used position that would be occupied by the building hit.

Due to these caches, the information stored when a photon hits a surface is increased with a new element so that the information stored is: position of the hit, the irradiance, the type of element hit and the position of the atlas where this element is placed. The splatting process does not change except that, when the splat is done in the facade photon map atlas, the blending is enabled to allow the photons of diferent street lights that hit a building to be stored in the photon map. Therefore, the facade photon map atlas is a dynamic structure even when no element changes in its cache. Furthermore, when a position of a cache is freed, the cache deletes the information stored in its structure an also deletes the region of the photon map atlas related to this position of the cache.

Algorithm 4 Pseudocode that shows how the grid structure and caches modify the rendering process.

```
1:  procedure UpdateCamera(Position pos)
2:      lights = Grid.getStreetLights(pos)
3:      for all l ∈ lights do
4:          if l ∈ FloorCache then
5:              cachePos = FloorCache.getAtlasPosition(l)
6:          else
7:              if  thenFloorCache.hasFreePosition()
8:                  cachePos = FloorCache.getFreePosition()
9:              else
10:                 cachePos = FloorCache.getLRUPosition()
11:                 FloorCache.freePosition(cachePos)
12:             end if
13:         end if
14:         photons = TracePhotons(l, cachePos)
15:         splatPhotons(photons)
16:         updateImprovedLightMaps()
17:     end for
18: end procedure
```

Algorithms 4 and 5 show this process. This structure provides the method a simple and efficient way to manage the photon map atlases. Thanks to the caches, photon map atlases dynamically change satisfying the requirements of the rendering every moment. The Figure 4.18 shows

Algorithm 5 Pseudocode that shows how the caches modify the hit function during the photon tracing process.

```
1: procedure ClosestHit(Photon p, Position pos, Energy e, FloorCachePosition fcPos)
2:     if phitsFloor then
3:         result.type = Floor
4:         result.cachePos = fcPos
5:     else if phitsFrontFacade then
6:         result.type = FrontFacade
7:         b = getBuildingHit(p)
8:         if b ∈ FrontCache then
9:             result.cachePos = FrontCache.getAtlasPosition(b)
10:        else
11:            if then FrontCache.hasFreePosition()
12:                result.cachePos = FrontCache.getFreePosition()
13:            else
14:                result.cachePos = FrontCache.getLRUPosition()
15:                FrontCache.freePosition(cachePos)
16:            end if
17:        end if
18:    else if phitsBackFacade then
19:        result.type = BackFacade
20:        b = getBuildingHit(p)
21:        if b ∈ BackCache then
22:            result.cachePos = BackCache.getAtlasPosition(b)
23:        else
24:            if then BackCache.hasFreePosition()
25:                result.cachePos = BackCache.getFreePosition()
26:            else
27:                result.cachePos = BackCache.getLRUPosition()
28:                BackCache.freePosition(cachePos)
29:            end if
30:        end if
31:    end if
32:    result.position = pos
33:    result.energy = e ∗ BRDF
34:    return result
35: end procedure
```

the photon map atlases after some rendering process.

4.5 Push-pull integration

The last part of our proposed method is the integration between the aerial and the close-view approaches. This part should allow the navigation through the scene using each method as needed. Furthermore, it will use the information generated in the photon mapping approach to increase the realism of the approximated part. In the next sections, we will introduce the solution proposed to satisfy these objectives in real time. Figure 4.9 shows this step in the overview of the technique.

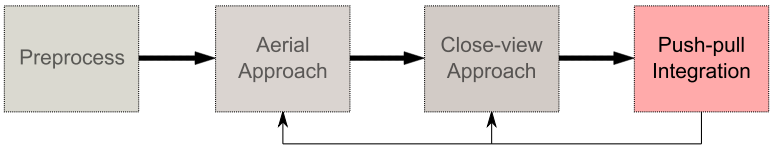


Figure 4.19: Section 4.5 represents the fourth box of the diagram.

4.5.1 Transferring information

One of the most important characteristics of our integration method is that it is inspired on the push-pull technique of hierarchical radiosity. The underlying idea of this technique is to use the information generated in both approaches to improve each other.

Transferring information from the aerial approach to photon mapping, as we mentioned before, consists on fixing the initial energy of the photons before being traced. The initial energy of the photons is set according to the direction of the photon and where this direction intersects the region influenced for the street light in the aerial approach. The position of this intersection and the approximation formula are used to compute this initial energy. Recall that the formula is given by:

$$\begin{aligned} I(x,y) &= A\cos^n \frac{r(x,y)}{r_{max}} && \text{if } r(x,y) \leq r_{max} \\ I(x,y) &= 0 && \text{if } r > r_{max} \end{aligned} \tag{4.17}$$

Transferring information in the other direction, from photon mapping to aerial approach, is more complex. The point of this information transfer must be to correct the approximation errors with more accurate information that takes into account the reflections, specular materials and other physical phenomenas. Passing information from photon mapping to the aerial approach means to transfer information to the aerial data structures. But

Figure 4.20: Example of the improved light maps (left floor light map and right front facade light map) after several transfers from the photon mapping approach.

one must be careful, as once new information is stored in the light map, it is impossible to differentiate the new information from the old one and this can be a source of problems. For this reason, it is better to create new structures to isolate old information from the new one.

The new structures are two new three-channel light maps: one for the floor and other for the facades. This light maps have three channels to be able to store the effect that the BRDF of the surfaces produces on photon bounces.

The improved floor light map is very similar to the original light map. Besides the three mentioned channels, the generation is different than in original one. Every time a photon mapping computation is done for a street light, a quad of size $2r_{max} \times 2r_{max}$ is rendered in the improved light map centered at the light position, as it is done in the original light map. The difference is that this quad is textured by downsampling the floor photon map instead of using the approximation formula. We can use the mipmapping technique to get the downsampled version of the texture atlas element.

The improved facade light map is a new data structure. In fact, there is one light map for the front facades and one for the back ones. This structure consists of a texture atlas with as many positions as building blocks there are in the city. For each of these elements, we use the cylindrical parametrization introduced in Section 4.2.1. Since we use the same parametrization, each time an element of the facade photon map atlas is generated or updated, we can downsample it via mipmapping and transfer it to the improved facade light map.

In Figure 4.20 it is shown an example of the the improved floor light map and the improved front facade light map. The use of mipmapping in the

generation of the improved light maps is a simple and efficient way to transform photon mapping results to light maps that can be directly used in the aerial approach. This process of transferring information from photon map to the improved light maps it is reflected in *updateImprovedLightMaps*() call in Algorithm 4.

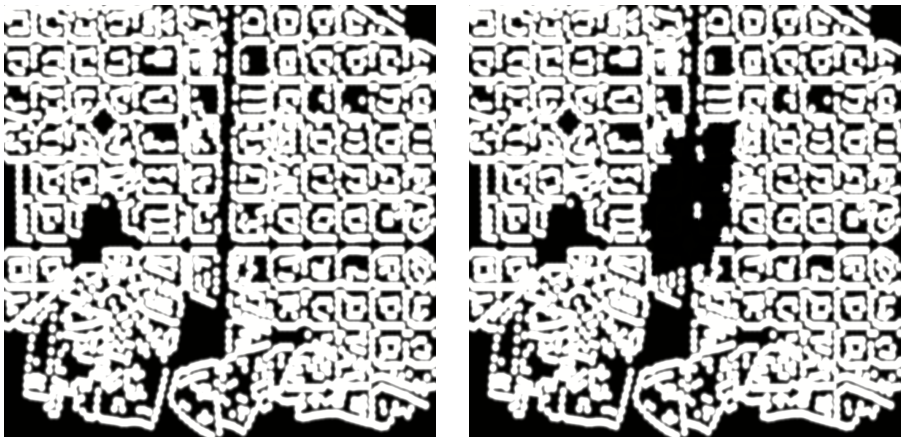


Figure 4.21: Example of a light map (left) and the resulting light map after removing the information of several street lights (right).

The last important step of the transferring method is based on the fact that, once an improved version of the aerial approach is obtained, it does not make sense to preserve this incorrect information any more. It was very useful before any correct computation because it was a fast approximation, but once a better one is calculated it should be deleted. As the original light map is generated by rendering textured quads for each street light, to remove the old information, we render the quad of the street light we desire to remove its texture weighted with negative values. In this way, as blending is enabled, we will subtract the contribution of the street light without removing the contribution of the neighboring street lights. In Figure 4.21 it is shown a light map which several street lights removed.

4.5.2 Final rendering

Once the information exchange between aerial and close-view approaches has been defined, the visualization of the final solution has to be redefined to use both approaches and their improved versions. The most important advantage of the proposed transferring method is the simplicity of its visualization.

Let $L = \{l_i\}$ the set of light sources in the scene. As each light source has a finite area of influence, let us assume each light source has an infinite

influence domain defined at point p as

$$l_i(p) = \begin{cases} F(p, q_i) & dist(p, q_i) \leq r_{max} \\ 0, & \text{otherwise} \end{cases}$$

where $F(p, q_i)$ is the function describing the illumination at point p by the i-th light located at q_i. Thus, the evaluation of the illumination at any point p in the city simply means evaluating

$$E_{total}(p) = \sum_i l_i(p)$$

When our algorithm starts, it initializes every light source $l_i(p)$ with the approximation computed with our aerial approach, i.e., $l_i(p) \approx I(p, q_i)$, where $I(x, y)$ is as defined in Eq. 4.17. This information is stored in our initial light map A, so $L = A$. From now on, we will add a superindex indicating the data structure containing the information about each light source, so $E_{total}(p) \approx \sum_i I(p, q_i) = \sum_i l_i^A(p)$.

During a close-view walkthrough, some of these lights are re-computed using photon mapping and stored in the cache. Our algorithm, at this point, removes them from A and stores them in the cache C as $l_i^C(p)$. Immediately, these light sources are also transferred to the improved light map RGB. However, as both maps (the cache C and the improved light map RGB) have different resolutions, we must downsample the high-resolution information in C. We label the resulting light distribution as $\hat{l}_i^{RGB}$. Thus, downsampling can be regarded as a function that transforms $l_i \xrightarrow{downsample} \hat{l}_i$, but, roughly speaking, the information contained in both light distributions is roughly the same (i.e., $l_i^C \approx \hat{l}_i^{RGB}$).

Computing the final illumination for every point simply means adding all contributions of all light sources that illuminate this point. For an aerial image, this means adding the contributions of the A and RGB light maps, as $L = A \cup RGB$:

$$E_{total}(p) = \sum_i l_i(p) = \sum_{i \in A} l_i^A(p) + \sum_{i \in RGB} l_i^{RGB}(p)$$

However, for street-level views, we might want to use the high-resolution information stored in the cache C, when available. Thus, the contribution of these lights would be computed as $\sum_{i \in C} l_i^C(p)$. However, this same contribution is already stored in our RGB map, so we must be careful not to count it twice. The solution is, for these lights only, to subtract the downsampled contribution and keep the high quality one. We do it as

$$E_{total}(p) = \sum_i l_i(p) = \sum_{i \in A} l_i^A(p) + \sum_{i \in RGB} l_i^{RGB}(p) + \sum_{i \in C} l_i^C(p) - \sum_{i \in C} \hat{l}_l^C(p)$$

if we recall that $l_i^{RGB}(p) = \hat{l}_i^C(p)$, this sum is equivalent to

$$E_{total}(p) = \sum_{i \in A} l_i^A(p) + \sum_{i \in RGB, i \notin C} l_i^{RGB}(p) + \sum_{i \in C} l_i^C(p) \tag{4.18}$$

which again provides the correct sum $\sum_i l_i(p)$. Observe that Eq. 4.18 is the most general, as it is valid for aerial views because $C = \emptyset$ and thus $L = A \cup RGB$ account for all the light sources. In the case of close views when we include the photon map, we are actually computing $L = A \cup RGB \cup C - \hat{C}$ (this last expression is valid thanks to the linear superposition principle).

This is a generalization of our scenario, but if we think about the floor and the facades, they both have their own photon maps and improved light maps, but the original light map is shared. So the energy of each part is computed independently by sharing the information of the light map.

The final render is very simple because we do not have to worry whether the geometry is far or near, as this work is, roughly speaking, done by the caches. The caches contain the more recently needed street lights, so we use the accurate photon map energy for them. For the other street lights we use the information of the light map or the improved light map according to whether they have been computed or not. So in the render the irradiance of each pixel will be computed as the summation of the irradiance stored in the light map, in the photon map and in the improved light map. The Algorithm 6 shows a pseudocode of the fragment shader that computes the final render.

Algorithm 6 Fragment shader's pseudocode for rendering the scene for near and far views using the integration technique of the close-view and aerial approaches.

```
1: procedure RENDER(Pixel p)
2:     if p ∈ Floor then
3:         color = texture2D(orthophoto, texCoord(p))
4:         irradianceLM = texture2D(lightMap, texCoord(p))
5:         irradianceLM+ = texture2D(floorLM+, texCoord(p))
6:         if p ∈ FloorPhotonMap then
7:             irradiancePM = texture2D(floorPM, texCoord(p))
8:             downsamplPM = texture2D(downfloorPM, texCoord(p))
9:         else
10:            irradiancePM = 0
11:        end if
12:    else if p ∈ Facade then
13:        color = texture2D(facadesTexture, texCoord(p))
14:        irradianceLM = texture2D(lightMap, texCoord(p))
15:        building = getBuilding(p)
16:        if p is a FrontFacade point then
17:            texCoordLM = getCoordinatesInLM(p)
18:            irradianceLM+ = texture2D(frontLM+, texCoordLM)
19:            if p ∈ FrontFacadePhotonMap then
20:                texCoordPM = getCoordinatesInAtlas(p)
21:                irradiancePM = texture2D(frontFacadePM, texCoordPM)
22:                downsamplPM = texture2D(downfrontPM, texCoordPM)
23:            else
24:                irradiancePM = 0
25:                downsamplPM = 0
26:            end if
27:        else if p is a BackFacade point then
28:            texCoordLM = getCoordinatesInLM(p)
29:            irradianceLM+ = texture2D(backLM+, texCoordLM)
30:            if p ∈ BackFacadePhotonMap then
31:                texCoordPM = getCoordinatesInAtlas(p)
32:                irradiancePM = texture2D(backFacadePM, texCoordPM)
33:                downsamplPM = texture2D(downbackPM, texCoordPM)
34:            else
35:                irradiancePM = 0
36:                downsamplPM = 0
37:            end if
38:        end if
39:    end if
40:    irradiance = irradianceLM + irradiancePM + irradianceLM+ −
   downsamplPM
41:    fragmentColor = color ∗ (ambientComponent + irradiance)
42:    return fragmentColor
43: end procedure
```

Chapter 5

Experimental Results and Discussion

5.1 Results

In this section, we present the results we have obtained with the proposed technique. First we will present the visual results and, after that, we will present a number of performance tests.

5.1.1 Rendering

The presented images show the behavior of the proposed technique in different situations: aerial views, close views and pedestrian views. Moreover, we show how the results change according to the approach used by the technique in each situation. During the renderings shown below, we have traced 75K photons from each light source when the close-view approach is used. We have organized the images shown to emulate a navigation through the scene.

First, Figure 5.1 shows the results of the initial aerial visualizations. In this situation, all the irradiance used to compute the illumination comes from the original light map. The images show the aerial approximation is good enough for distant views when no realistic light information is available.

Figure 5.2 shows the moment the close-view approach is triggered. This happens, as we mentioned before, when the user is near the floor. Note indirect illumination effects on these images.

Figures 5.3 and 5.4 shows images of pedestrian views. We can see the details of the illumination shapes and indirect illumination effects as color bleeding. Furthermore, we can see how the approximated solution is still applied for the far geometry while the realistic solution is applied to the closest ones. Both of them coexist in a natural way as we can see in the images.

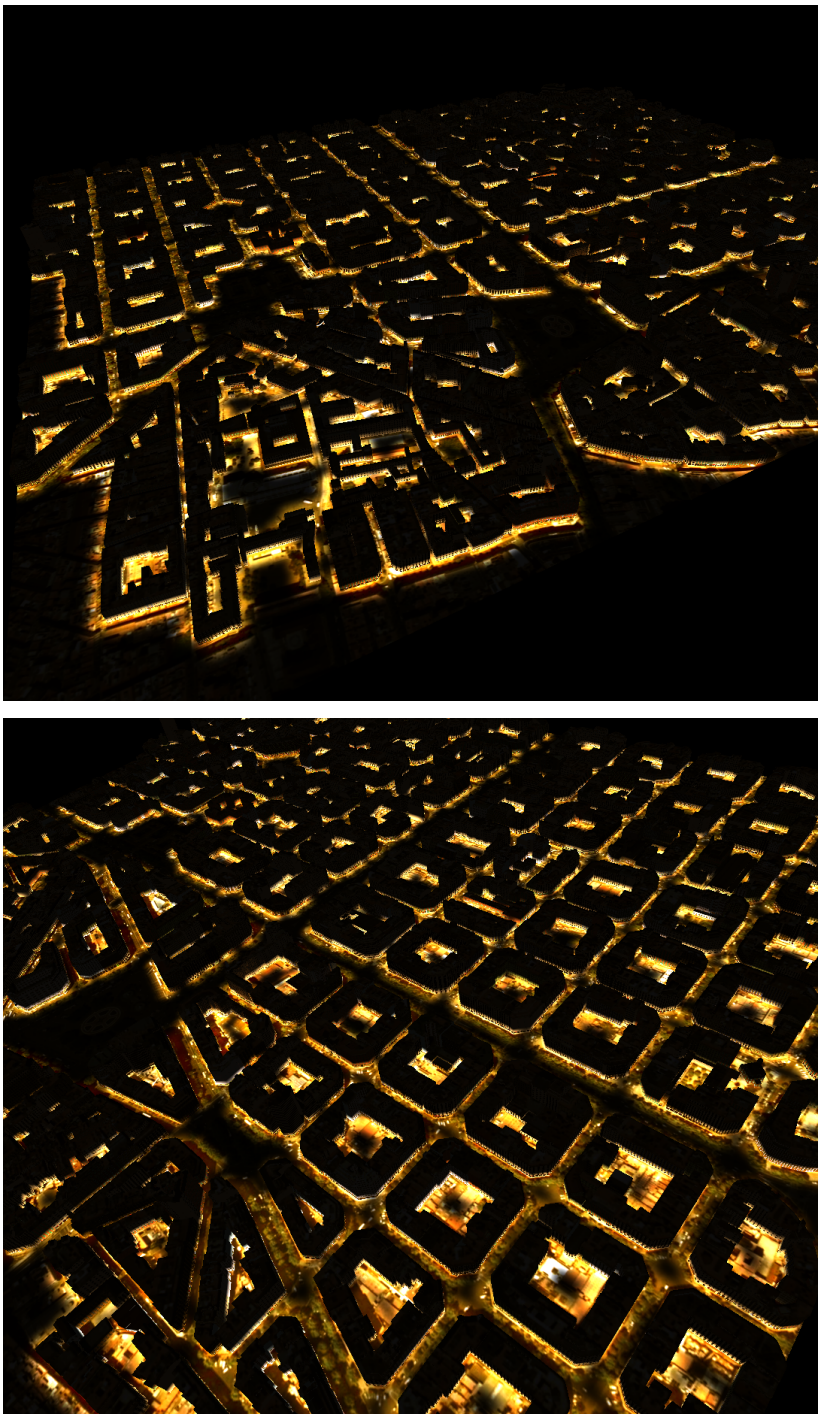


Figure 5.1: Rendering results of aerial views when all the information to compute illumination comes from the original lightmap.

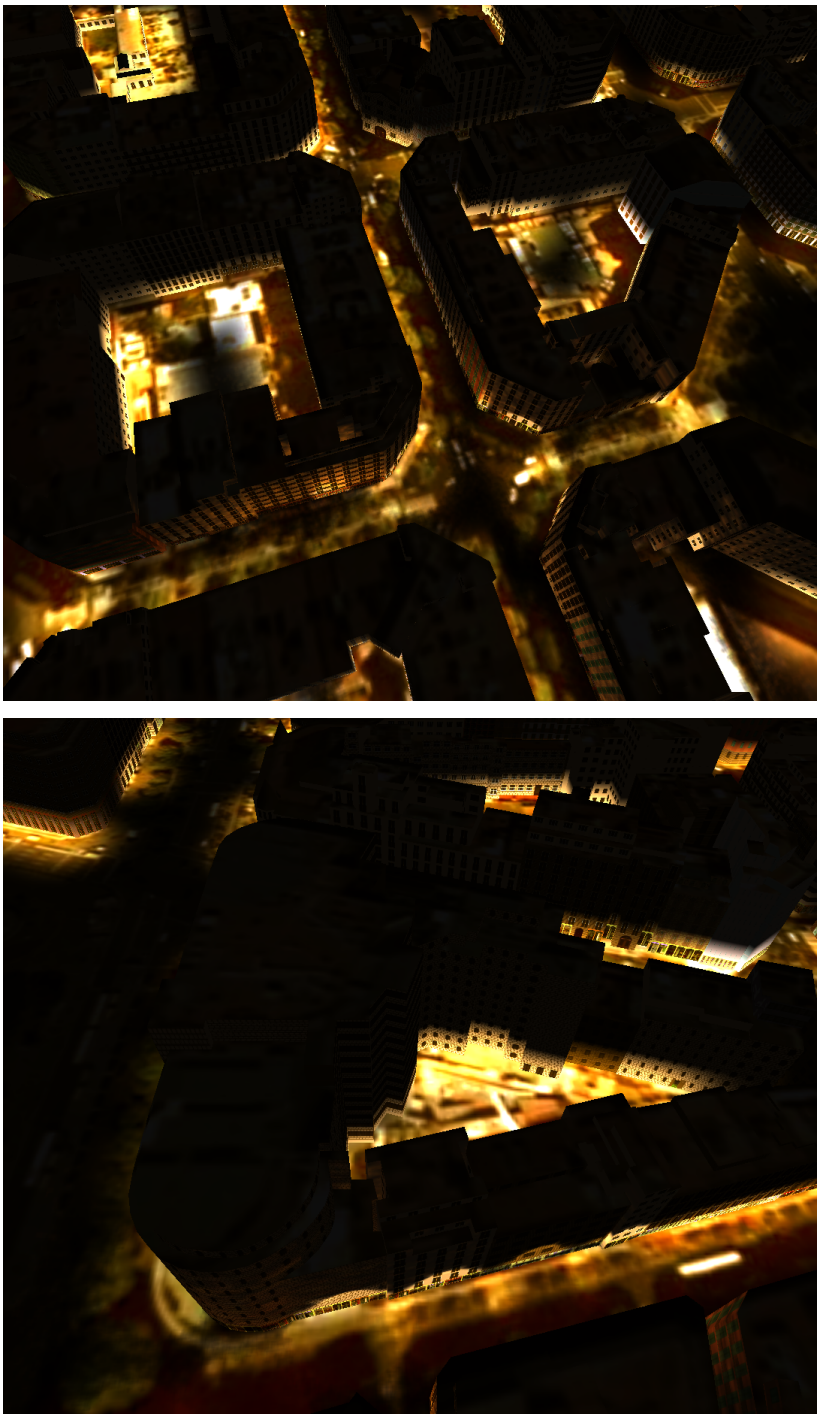


Figure 5.2: Images obtained when the close-view approach is triggered. Note how the realistic solution is applied to the closest zone while the approximated solution is for the rest.

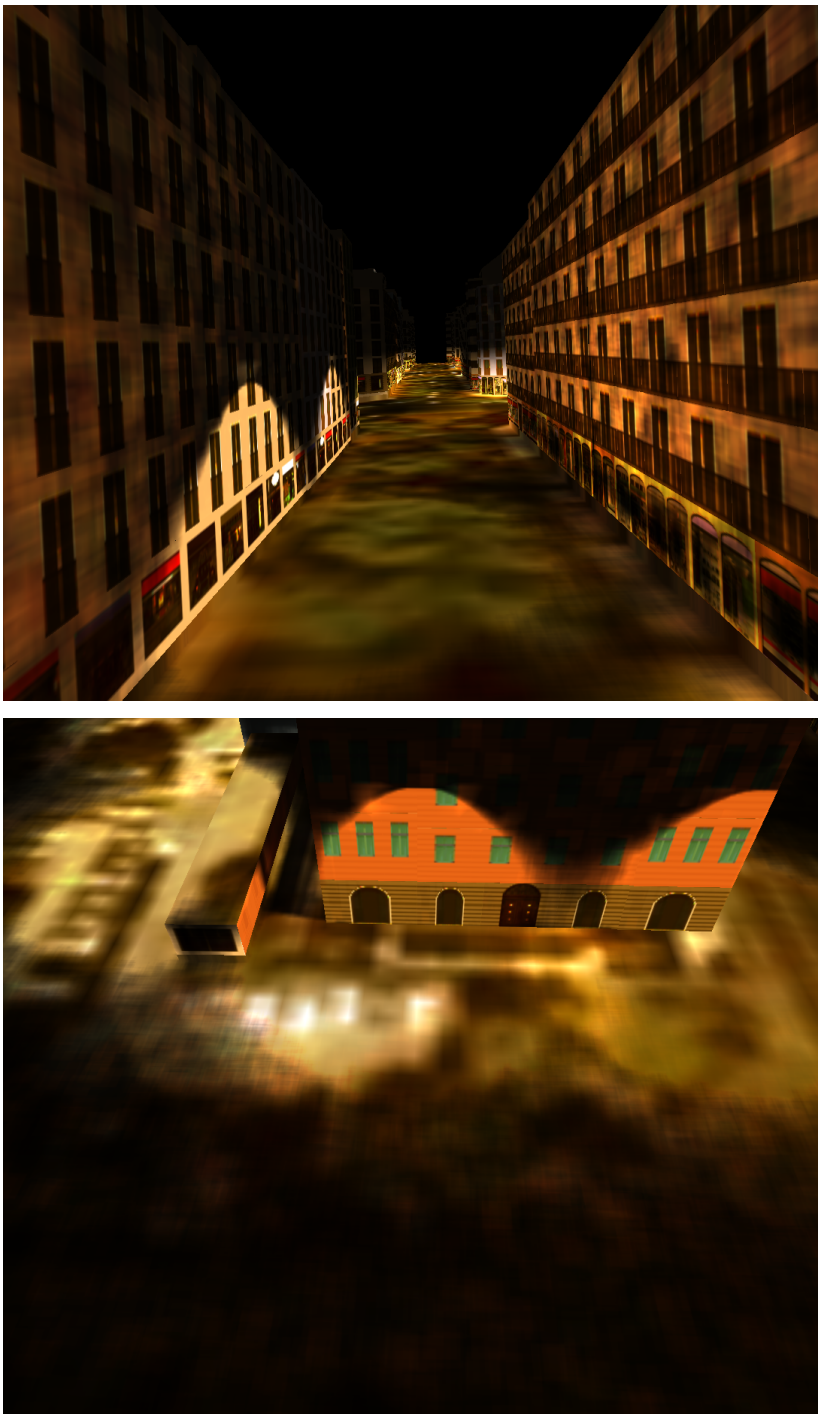


Figure 5.3: Renderings from very close distances. Top: effect of indirect illumination. Bottom: note how the illumination improves in a realistic way with using photon mapping. The poor resolution of the orthophoto in our test dataset becomes evident.

Figure 5.4: Close-view solutions compute several indirect illumination phenomenas like color bleeding, which is very noticeable on these images.

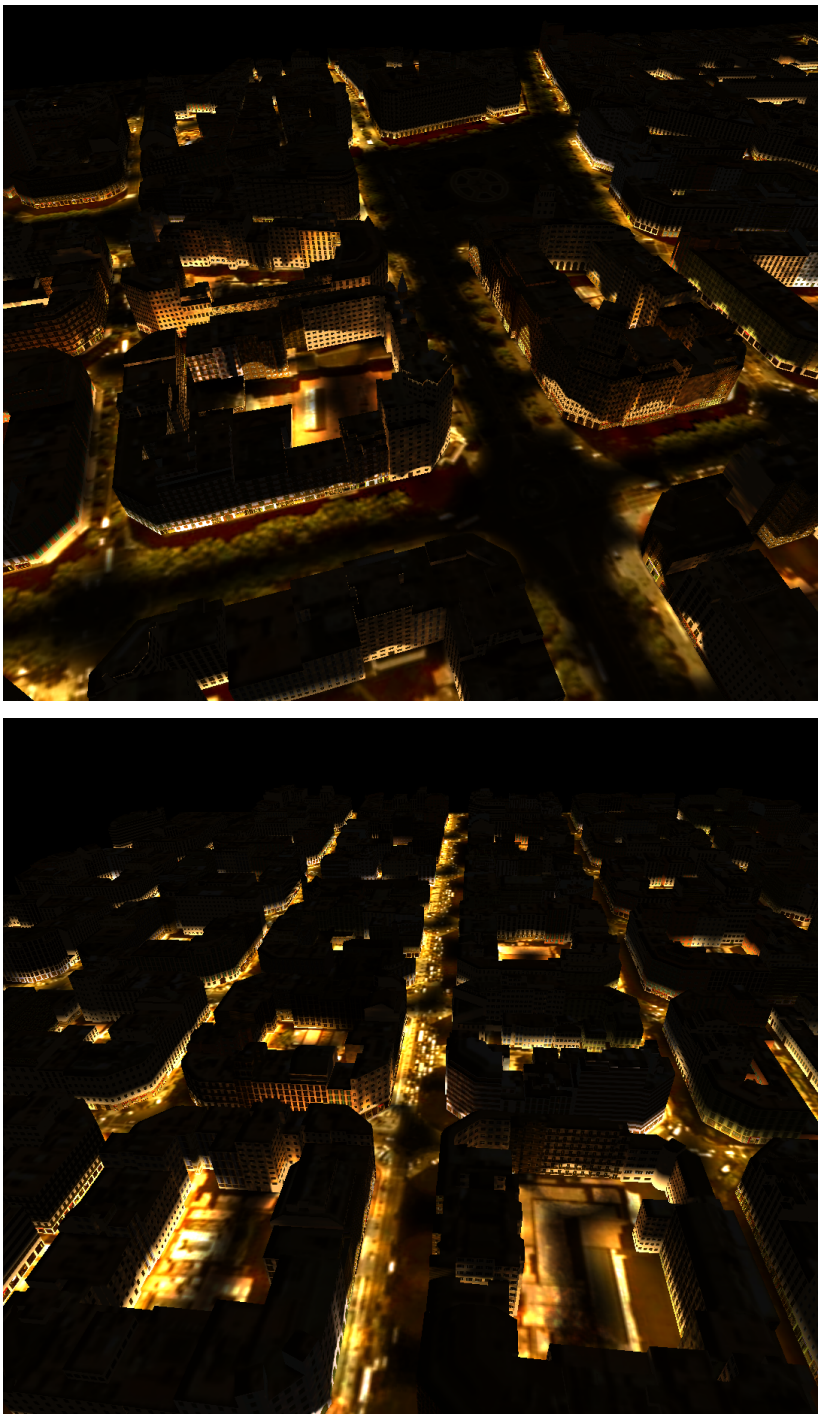


Figure 5.5: The push pull integration improves the information of the approximated solution. This improvement can be seen on these images where indirect illumination is shown by the approximated solution.

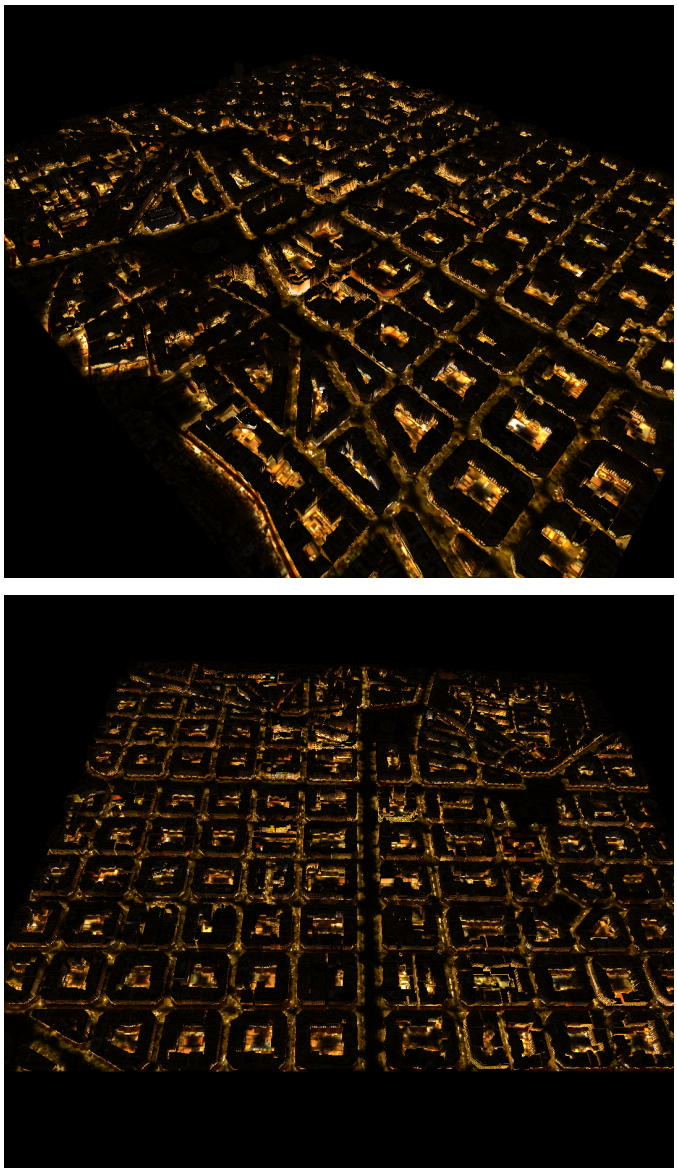


Figure 5.6: Inspection of large zones of the scene implies the improvement of the original light map. In these images we show how the push pull integration can provide indirect illumination to the whole city.

After close inspections, we can see the effects of the push-pull integration on the images. Figure 5.5 shows how this approach improves the original light map incorporating realistic information of the close view approach. This allows to see indirect illumination phenomena for aerial views and improve the shapes of the illumination. Moreover, like in the closest views the coexistence of the approximated and realistic solutions is very harmonious.

The improvement process of the light map continues along the visualization. More close inspections implies more improvement of the light map. Related with that, Figure 5.6 shows distant aerial views after doing the close inspection for the whole city. In this image, we can see how the distant views improve their quality during the render.

5.1.2 Performance tests

One important initial goal is related with the performance of the technique. For this reason, we did several performance tests to analyze the viability of the technique. For all the tests we used a 3.10 GHz quad core Intel Core i5-2400 CPU with 4 GB memory running Microsoft Windows 7 64-bits and a NVIDIA GeForce GTX 460 GPU with 1 GB memory.

First, we studied the performance of the close-views approach. This approach uses a variation of photon mapping technique, so this test can be an evaluation of our photon mapping variation. The test consists of rendering the same view several times and each time changing the number of photons comparing the time cost and the quality of the image. To measure the quality of an image we used a very simple test, we generated the image using three million photons (See Figure 5.7) and we measured the error with respect to this image computing the mean difference between their pixels.

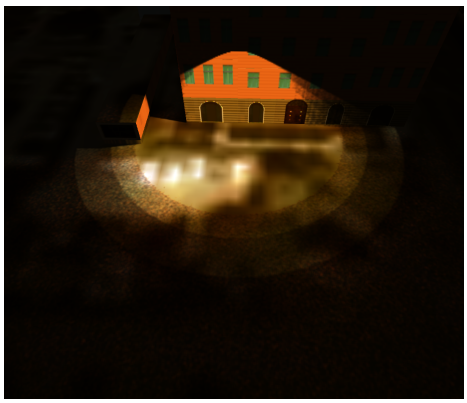


Figure 5.7: Target image on the quality test.

Figure 5.8 shows the results of this test. We can see that the time cost increases linearly with the number of photons used while the quality error decreases in an exponential way. We can see in the chart that from emitting 50K photons the error obtained is acceptable and the performance is very high. When the measured error is under 1% the time cost is too high. The performance measure becomes unstable after emitting more than 1.9M photons. There are several reasons that contribute to this behaviour, but we believe that the main one is an extreme workload on the Optix engine. We base our hypothesis in the several crashes we experienced for high workloads. Analysing the chart, we arrived to the conclusion that tracing 75K photons we obtain an image with enough quality and with a low computational cost. For this reason, the rendering experiments (See Section 5.1.1) and the following performance tests use this value as the number of emitted photons.

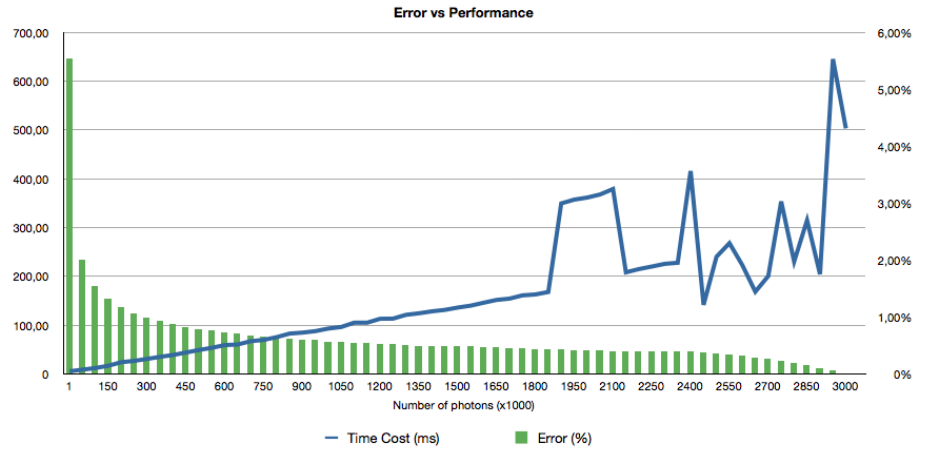


Figure 5.8: Performance results compared to the error estimation obtained.

After studying the performance of the photon mapping algorithm, it is important to study the performance of the whole technique. Table 5.1 shows the time cost of each of the steps of the algorithm in two situations: tracing 75K and 1M photons in a navigation where only one light is computed each time. First we computed the time cost of generating the light map. After that, we divided the computation cost in the operations that involve the cache, the tracing step, the splatting step, the push-pull integration and the generation of the mipmaps. After that is computed the drawing cost.

Emitted photons	Preprocess Lightmap	Computation					Render Draw	TOTAL
		Cache	Trace	Splat	Push-pull	Mipmap		
75K	5.43	0.12	10.80	1.80	0.87	0.23	0.82	20.07
1M	6.02	0.20	105.04	17.69	7.77	0.25	1.14	138.11

Table 5.1: This table show the time cost in ms of each of the steps of the technique. Preprocess group is only computed once in all the visualization. Computation steps and render is the mean of their time cost per frame during a navigation where only one light is computed each frame.

As expected, the steps depending on the number of photons (tracing and splatting) linearly increased their computational time, while the other steps remained almost constant. The deviation comes from the fact that increasing the number of emitted photons increases the number of buildings hit, and, therefore, the number of cache movements. Also, the push-pull step requires generating an intermediate texture, which depends linearly on the number of the emitted photons.

Finally, a frame rate test during a navigation was performed. The test was done on a city model with 166 blocks, 274K vertices and 141K triangles. For each street light simulated with the close-view approach 75K photons

were traced. During the experiment, to check the performance under hard conditions, we computed the lights of the grid cell and its neighbors. This variation gives more visual quality to the user because the action radius of the close-view approach is higher. For this example, the navigation starts from an aerial view to a close view with fast and abrupt movements, including a pedestrian view and a return to the far view. Figure 5.9 shows the frame rate during this navigation.

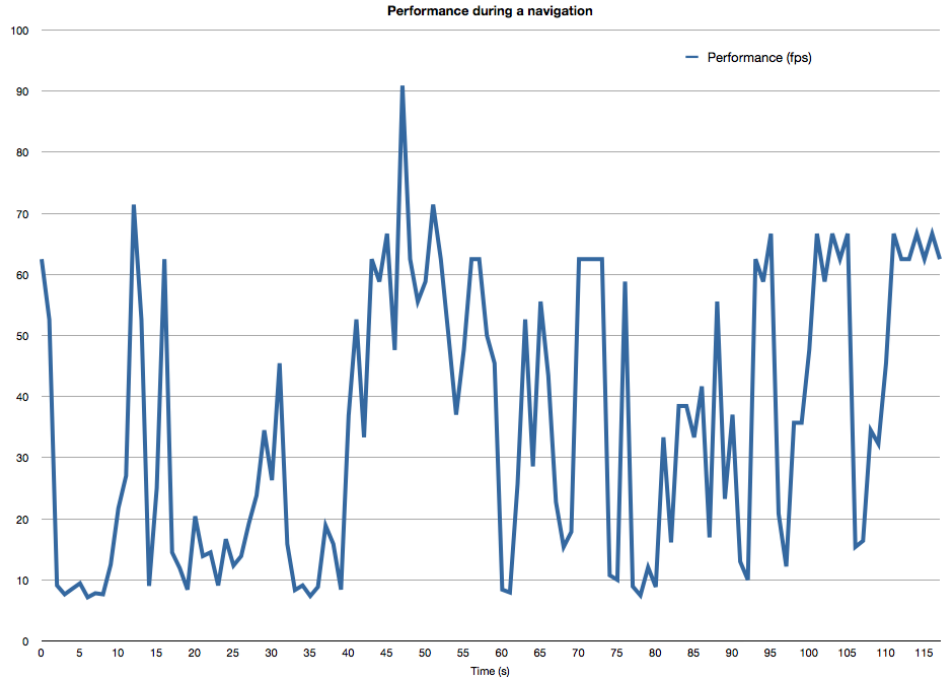


Figure 5.9: Performance in fps during a navigation along the Barcelona city model.

Analyzing the performance chart, we can appreciate that although the technique works in real time frame rates there are cases where the performance falls under 10 fps. This situations correspond to when the whole cache has to be updated. For example at the beginning and during the fasts navigations at the height where the close-view approach starts to be used (Figure 5.9: from 2s to 8s, from 17s to 25s and from 32s to 40s). The movement of the camera is too high and cache information must be computed every frame. During close inspections, the frame rate is over 60 fps but it drops occasionally in big updates of the cache (Figure 5.9: seconds 14, 60, 74, 92 and 97). This is caused for the hard conditions of extending the region where the close-view approach is used. During aerial views (Figure 5.9: from 40s to 60s) the frame rates are really high exceeding the 90 fps.

Should a more constant frame rate be desirable, one can fix an upper

bound to the number of cache updates per frame. This would result in a more fluid navigation at the expense of using approximate radiance for middle-range street lights.

5.2 Discussion

The proposed method offers a solution that provides an approximation good enough for far views and a physically realistic solution to compute the global illumination of near views. Furthermore, the correct integration between both of them, thanks to push-pull-based integration technique, provides a unified solution with a smooth transition between both methods.

The light map generated in the aerial approach is a very fast solution as we can see in Table 5.1. It provides a first rough computation with enough quality (See Figure 5.1 5.10). Direct illumination is approximated correctly and the general look of the scene is realistic enough.

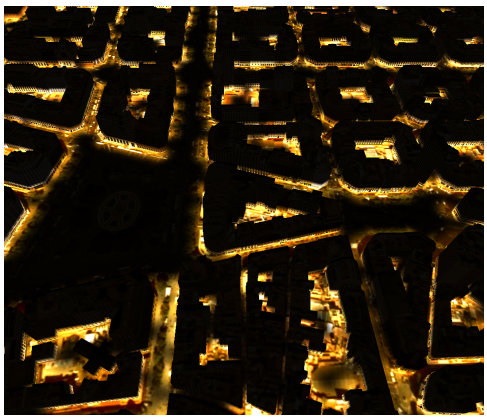


Figure 5.10: Direct illumination is approximated correctly and the general look of the scene is realistic.

The close-view approach is a fast approximation that produces physically realistic images. Our variation of photon mapping achieves high frame rates with enough quality (See Figures 5.2 5.3 5.4 5.11 and Table 5.1). The cache solution allows us to use photon mapping with high quality and without prohibitive memory costs. Subdividing the scene with a regular grid and using it to know which street lights should be computed at every moment is a fast solution that provides good results. In previous section we have seen its limitations under extreme situations. Extending the close-view approach to the neighbor of the current cell of the grid converts the cache update in the bottleneck of the technique (See Figure 5.9).

This problem can be solved with some kind of frustum culling technique before computing the street lights. The current solution computes the information of street lights that do not strictly provide any information to the

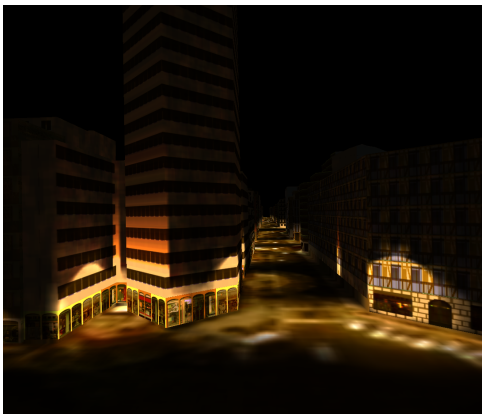


Figure 5.11: The close-view approach result has a harmonious integration with the approximation computed to the far views.

current view. Another solution to this problem could be based on progressive computation. For example, computing first the current grid cell and postponing the neighbors to the subsequent frames. These solutions would smooth the drops in the performance.

The absence of indirect illumination in the aerial approach is solved progressively through our push-pull integration. This method, inspired on hierarchical radiosity technique, is a clever solution that provides the aerial view with physically realistic information. The information is updated while the user navigates along the scene without the user can realize of it: only when the user returns to aerial views will notice this improvement. The result is a very high quality images for far and medium distances in high frame rates (See Figures 5.5 5.6 5.9 5.12 and Table 5.1).

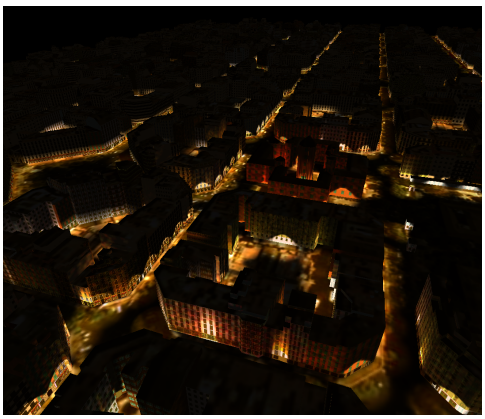


Figure 5.12: After the push pull integration the quality of the medium distance views is really high.

Another important property of the technique is that each data structure consumes small computational times during rendering and, provide the needed quality in each situation. The fact of isolating approximated and realistic computations in different structures, allows an efficient and secure change from one to the other. Furthermore, this separation allows us to use the generalized rendering formula we have introduced in Section 4.5.2 (See Figure 5.13).

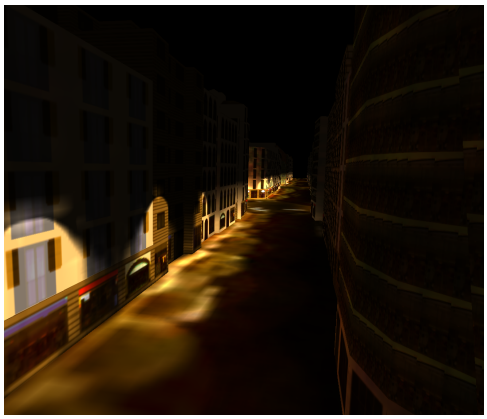


Figure 5.13: The generalized rendering formula simplifies the rendering process with results of high quality.

This generalized formula makes it possible to use the information of the photon maps stored in the caches even for far views. Although it can be regarded as a flaw, in fact it is an advantage. As we have mentioned in Section 4.5.2, if we have for a region information in the cache, its information in the improved light map is not used since the formula subtracts it. As we use mipmapping on rendering and to generate the improved light map, while the user goes far away the render will use the necessary mipmap level at each moment. When the user is far away enough, the same mipmap level that was used to generate the improved light map of that region will be used. So, between the far away aerial view and the closer view the illumination of the last closest visited region would be computed with higher detailed information and a smooth transition.

One element that reduces the quality of the obtained images is the low quality of the city textures. They have synthetic and repetitive appearance. Even worse, the low quality of the orthophoto also influences in the rendering quality. It has low resolution and there are elements (cars, trees, etc) projected on it. Moreover, the orthophoto contains shadows of the buildings which introduce a red artifact in small regions after applying histogram matching technique. Using textures and orthophotos of a higher quality would improve the results considerably.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The main goal of this master thesis is to compute the nocturnal illumination of an urban scene using global illumination techniques and to visualize it in real time. After the studying the state of the art in photon mapping and urban rendering, we have observed that global illumination in urban scenes has been briefly addressed in the literature.

We have presented a technique that computes the nocturnal illumination of a city in real time. Our technique uses a variation of photon mapping to simulate global illumination effects and achieve physically realistic results. As the problem of computing the illumination of a city during the night involves the computation of a high number of street lights, we have divided the problem in two cases.

First, we have presented an aerial approach that computes a fast approximation of the illumination of the city. After that, we have presented a close-view approach that uses an optimized photon mapping technique to compute the global illumination of the nearest subset of street lights. We have presented also a solution to seamlessly integrate both methods, inspired in push-pull that uses the information of both approaches to improve each other. Finally the rendering is done with a generalized formula that combines the three techniques obtaining high quality images. Furthermore, our proposed technique achieves real time frame rates.

6.2 Future Work

There are some aspects and limitations that we would like to work on in our future research. First, we would like to improve the algorithm that places the street lights to detect wide streets and place street lights in the

middle of the street at this happens in the avenues. Furthermore, we would like to render a model with a sphere and a halo placed in the street lights positions to improve the realism of the scene. Moreover, we would like to incorporate the possibility of simulating real street lamps using IES stands for Illuminating Engineering Society.

Although we mentioned in the previous chapter that the quality of the buildings texture and orthophoto should be improved, we would also like to study more the histogram matching algorithm to avoid the red artifacts produced in small regions of the orthophoto.

Computing the illumination of specular surfaces is now working in the thesis. We would like to extend it, computing the visualization of specular surfaces. This would increase the realism of the scene significantly because it would improve the rendering of windows, mirrors and glass buildings.

Finally we would like to extend our method incorporating typical urban elements, such as trees and cars, and studying their interaction with light in the context of nocturnal lighting.

Bibliography

[1] Timo Aila and Samuli Laine. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 145–149, New York, NY, USA, 2009. ACM.

[2] C. Andújar, P. Brunet, A. Chica, and I. Navazo. Visualization of large-scale urban models through multi-level relief impostors. *Computer Graphics Forum*, 29(8):2456–2468, 2010.

[3] Oscar Argudo, Carlos Andújar, and Gustavo Patow. Interactive rendering of urban models with global illumination. In *Computer Graphics International*, Bournemouth University, United Kingdom, June 2012.

[4] Donald K. Carter, Stephen Quick, Stefani Danes, Elise Gatti, and Karen Branick. Led street light research project. Technical report, Carnegie Mellon University and Remaking Cities Institute, Pittsburgh, PA, USA, 2011.

[5] R. Chang, T. Butkiewicz, C. Ziemkiewicz, Z. Wartell, W. Ribarsky, and N. Pollard. Legible simplification of textured urban models. *Computer Graphics and Applications, IEEE*, 28(3):27 –36, may-june 2008.

[6] P. Cignoni, M. Di Benedetto, F. Ganovelli, E. Gobbetti, F. Marton, and R. Scopigno. Ray-casted blockmaps for large urban models visualization. *Computer Graphics Forum*, 26(3):405–413, 2007.

[7] M. Cohen, J. Wallace, J. Radiosity, In Artificial, Life Iii, C. G. Langton, and Ed. Addison-wesley. Radiosity and realistic image synthesis, 1993.

[8] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 203–231, New York, NY, USA, 2005. ACM.

[9] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 93–100, New York, NY, USA, 2006. ACM.

[10] M. Di Benetto, P. Cignoni, F. Ganovelli, E. Gobbetti, F. Marton, and R. Scopigno. Interactive remove exploration of massive cityscapes. In *Symposium on Virtual Reality, Archeology and Cultural Heritage VAST*, 2009.

[11] Jürgen Döllner and Henrik Buchholz. Continuous level-of-detail modeling of buildings in 3d city models. In *Proceedings of the 13th annual ACM international workshop on Geographic information systems*, GIS '05, pages 173–181, New York, NY, USA, 2005. ACM.

[12] Jan-Henrik Haunert and Alexander Wolff. Optimal and topologically safe simplification of building footprints. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, pages 192–201, New York, NY, USA, 2010. ACM.

[13] Robert Herzog, Vlastimil Havran, Shinichi Kinuwaki, Karol Myszkowski, and Hans-Peter Seidel. Global illumination using photon ray splatting. *Computer Graphics Forum*, 26(3):503–513, 2007.

[14] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, London, UK, UK, 1996. Springer-Verlag.

[15] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[16] H.W (organizer) Jensen, P.H Christensen, and Frank Suykens. A practical guide to global illumination using photon mapping. In *ACM SIGGRAPH 2001 Course Notes*, volume 38 of *SIGGRAPH '01*, pages 60–90, Los Angeles,USA, August 2001. ACM.

[17] Stefan Jeschke and Michael Wimmer. Textured depth meshes for real-time rendering of arbitrary scenes. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 181–190, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[18] Fabien Lavignotte and Mathias Paulin. Scalable photon splatting for global illumination. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '03, pages 203–ff, New York, NY, USA, 2003. ACM.

[19] Paulo W. C. Maciel and Peter Shirley. Visual navigation of large environments using textured clusters. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pages 95–ff., New York, NY, USA, 1995. ACM.

[20] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 77–89, New York, NY, USA, 2009. ACM.

[21] Tania Pouli, Douglas W. Cunningham, and Erik Reinhard. A survey of image statistics relevant to computer graphics. *Comput. Graph. Forum*, pages 1761–1788, 2011.

[22] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, pages 41–50. Eurographics Association, 2003.

[23] François Sillion, George Drettakis, and Benoit Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum*, 16(3):C207–C218, 1997.

[24] Wolfgang Stürzlinger and Rui Bastos. Interactive rendering of globally illuminated glossy scenes. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97*, pages 93–102, London, UK, UK, 1997. Springer-Verlag.

[25] C. A. Vanegas, D. G. Aliaga, P. Wonka, P. Müller, P. Waddell, and B. Watson. Modelling the appearance and behaviour of urban spaces. *Computer Graphics Forum*, 29(1):25–42, 2010.

[26] Ingo Wald, Philipp Slusallek, Carsten Benthin, and Markus Wagner. Interactive rendering with coherent ray tracing. *Computer Graphics Forum*, 20(3):153–165, 2001.

[27] M. Wimmer, P. Wonka, and F. Sillion. Point-based impostors for real-time visualization. In *Proc. of the Eurographics Workshop on Rendering 2001*, pages 163–176, London, United Kingdom, 2001.

[28] Ling Yang, Liqiang Zhang, Jingtao Ma, Jinghan Xie, and Liu Liu. Interactive visualization of multi-resolution urban building models considering spatial cognition. *International Journal of Geographical Information Science*, 25(1):5–24, 2011.